HyDRA - Hybrid workflow Design Recommender Architecture

Kamran Soomro

University of the West of England
Faculty of Environment & Technology

This thesis is submitted for the degree of
Doctor of Philosophy

March, 2016

Acknowledgements

I would like to thank my supervision team Prof Richard McClatchey and Dr Kamran Munir for their dedication and hard work in guiding me throughout this PhD process and for having faith in me and encouraging me when I most needed it. Also of particular note are Dr Zaheer Khan, Dr Ashiq Anjum, Dr Peter Bloodsworth, Dr Mohammad Odeh and Mr David Ludlow for their support and contributions. I would also like to thank my wife and my parents for bearing with me throughout this long and painstaking process. Without their love and patience I would not have been able to succeed. Finally, I would also like to thank my PhD examiners, Prof Andrew Jones and Dr Stewart Green for their time, patience and guidance.

Contents

[File Creation Timestamp: 24/02/2016, 14:13]

List of Figures

List of Tables

Glossary

Abstract component	a workflow component that is not executable.		ix, 57, 71
Abstract workflow	a workflow that contains at least one abstract component.		57, 63
.air	File format used by the AIR tools		101
Automated workflow systems	workflow systems that automatically generate workflows from user-provided descriptions		14
Bias field	a low-frequency, very smooth signal that corrupts Magnetic Resonance Image (MRI) images, particularly those produced by very old MRI machines		11
Biomarker	a substance used as an indicator of a biological process. Also see *Morphological signature*		3
Concatenate	concatenate operation between two sets	$\oplus$	56
Concrete component	a workflow component that is executable.		57
Concrete workflow	a workflow that contains no abstract components.		57
$\mathbf{D_m}$	system suggests multiple actions at one time		26
$\mathbf{D_s}$	system suggests only action at a time		26
Description-Logic	a family of formal knowledge representation languages.		xii, 55
Difference	subtraction operation between two sets	$-$	56
Disjoint relationship	a mutually exclusive relationship between two DL concepts		56
e-infrastructure	see *e-Science infrastructure*.		x, 1, 4

e-Science	computationally intensive science that is carried out in highly distributed network environments		x, 1–3
e-Science infrastructure	infrastructures that support e-Science		ix, 1, 3
Graph-based workflow modelling	a modelling technique in which workflows are visually expressed using graph-based structures		14, 15
Intersection	an intersection between two concepts in a DL specifies all individuals that belong to both concepts	$\sqcap$	56
Language-based workflow modelling	a modelling technique in which workflows are expressed using text-based workflow representation languages		14
Morphological signature	a structural signature of a biological process. Also see *Biomarker*		3
Magnetic Resonance Imaging	a medical imaging technique to visualise detailed internal anatomical structures		ix, xiii
neuGRID	an e-infrastructure project to allow neuroscientists to analyse MRI scans for the detection of biomarkers for Alzheimer's disease		3
Neurodegenerative disease	a disease in which neurons are progressively damaged		2, 3, 11, 115
Neuroimaging	the use of various techniques to image the structure and function of the brain		3, 11, 67, 91
$\mathbf{R_f}$	system ranks suggestions according to frequency		26
$\mathbf{R_h}$	system ranks suggestions based on heuristics		26
Recommender system	a system designed to recommend items of interests to users		4, 5, 16, 19, 22, 30, 35, 116, 118
registration	spatial normalisation in which the voxels of an image are matched to a template image first		11–14, 48, 57, 67, 93, 112
$\mathbf{S_p}$	system employs pattern-based suggestion generation strategy		26

S_s	system employs semantics-based suggestion generation strategy		26
segmentation	a process in which different brain structures are delineated on a brain image		66–68, 98
subsumed	see *Subsumption relationship*		56, 75, 77
Subsumption relationship	a subsumption relationship between two concepts in a DL states that one concept is more specific than another concept	$\sqsubseteq$	xi, 56, 57, 94, 112
Transitive dependency	a dependency that exists between the output and input of a component. It specifies whether semantics can be propagated across that component or not	σ	59, 65, 69, 70, 75
Union	a union between two concepts in a DL specifies all individual that belong to either one of them	$\sqcup$	56
User-directed workflow systems	workflow systems that require users to directly edit and design workflows		14, 15
Voxel	a volume element. Analogous to a pixel for a 3D image.		11

Acronyms

ADNI	Alzheimer's Disease Neuroimaging Initiative	3
BET	Brain Extraction Tool	66
BPEL	Business Process Execution Language	16
CAT	Composition Analysis Tool	21, 22, 24–26, 34, 77
CBR	Case-Based Reasoning	22, 26
CME	Community Modelling Environment	21
CSR	Component Suggestion Request	45
CSRH	Component Suggestion Request Handler	45
DAG	Directed Acyclic Graph	18
DAML+OIL	DARPA Markup Language+Ontology Inference Layer	27, 29
DL	Description-Logic	55–57, 65
EGEE	Enabling Grids for E-sciencE	1
FLIRT	FMRIB's Linear Image Registration Tool	12, 57, 66–71, 93, 98, 103
FSL	FMRIB Software Library	3, 57, 66
ISO	International Organisation for Standardisation	16

LONI	Laboratory of Neuro Imaging	i, 15, 16, 32, 95, 107
MAP	Mean Average Precision	87
MINC	Medical Image NetCDF	3
MRI	Magnetic Resonance Image	ix, x, 3, 4, 45, 49, 57–60, 66, 67, 71, 73, 74, 91, 93, 98–101, 103, 115
MRR	Mean Reciprocal Rank	84, 87, 95, 102, 107–111, 113
NIHPD	US-NIH MRI Study of Normal Brain Development	3
OWL	Web Ontology Language	27–30, 35
RDF	Resource Description Framework	27, 28
RDFS	RDF Schema	27, 29
ROI	region of interest	11–13
SA	Semantic Analyser	45
SBE	Suggestion Building Engine	45
SCEC	Southern California Earthquake Center	21
SPM	Statistical Parametric Mapping	3
SRB	Suggestion Request Broker	45, 47
SWRL	Semantic Web Rule Language	28, 30
UAM	User Action Monitor	45

Abstract

Workflows are a way to describe a series of computations on raw e-Science data. These data may be MRI brain scans, data from a high energy physics detector or metric data from an earth observation project. In order to derive meaningful knowledge from the data, it must be processed and analysed. Workflows have emerged as the principle mechanism for describing and enacting complex e-Science analyses on distributed infrastructures such as grids. Scientific users face a number of challenges when designing workflows. These challenges include selecting appropriate components for their tasks, specifying dependencies between them and selecting appropriate parameter values. These tasks become especially challenging as workflows become increasingly large. For example, the CIVET workflow consists of up to 108 components. Building the workflow by hand and specifying all the links can become quite cumbersome for scientific users.

Traditionally, recommender systems have been employed to assist users in such time-consuming and tedious tasks. One of the techniques used by recommender systems has been to predict what the user is attempting to do using a variety of techniques. These techniques include using workflow semantics on the one hand and historical usage patterns on the other. Semantics-based systems attempt to infer a user's intentions based on the available semantics. Pattern-based systems attempt to extract usage patterns from previously-constructed workflows and match those patterns to the workflow under construction. The use of historical patterns adds dynamism to the suggestions as the system can learn and adapt with "experience". However, in cases where there are no previous patterns to draw upon, pattern-based systems fail to perform. Semantics-based systems, on the other hand infer from static information, so they always have something to draw upon. However, that information first has to be encoded into the semantic repository for the system to draw upon it, which is a time-consuming and tedious task in itself. Moreover, semantics-based systems do not learn and adapt with experience. Both approaches have distinct, but complementary features and drawbacks. By combining the two approaches, the drawbacks of each approach can be addressed.

This thesis presents HyDRA, a novel hybrid framework that combines frequent usage patterns and workflow semantics to generate suggestions. The functions performed by the framework include; a) extracting frequent functional usage patterns; b) identifying the semantics of unknown components; and c) generating accurate and meaningful suggestions. Challenges to mining frequent patterns include ensuring that meaningful and useful patterns are extracted. For this purpose only patterns that occur above a minimum frequency threshold are mined. Moreover, instead of just groups of specific components, the pattern mining algorithm takes into account workflow component semantics. This allows the system to identify different types of components that perform a single composite function. One of the challenges in maintaining a semantic repository is to keep the repository up-to-date. This involves identifying new items and inferring their semantics. In this regard, a minor contribution of this research is a semantic inference engine that is responsible for function b). This engine also uses pre-defined workflow component semantics to infer new semantic properties and generate more accurate suggestions. The overall suggestion generation algorithm is also presented.

HyDRA has been evaluated using workflows from the Laboratory of Neuro Imaging (LONI) repository. These workflows have been chosen for their structural and functional characteristics that help

to evaluate the framework in different scenarios. The system is also compared with another existing pattern-based system to show a clear improvement in the accuracy of the suggestions generated.

[File Creation Timestamp: 24/02/2016, 14:13]

CHAPTER 1

Introduction

Recently scientific projects and applications are being increasingly characterised by their large-scale computation and data storage requirements. In order to meet these computation and data storage needs, scientific research is generally carried out using large-scale distributed computing infrastructures. This type of scientific research is generally termed as *e-Science*, and the infrastructures that support these endeavours are termed as *e-Science infrastructures*. It has been described as being "...about global collaboration in key areas of science and the next generation of infrastructure that will enable it."[1] e-Science spans domains from the Arts and Humanities to Physical Sciences and Engineering.

An e-Science infrastructure generally consists of a set of distributed heterogeneous computing sites. Such resources are commonly located within different institutions and may be geographically distributed around the world and connected via the internet. Each site may consist of a set of networked computing, storage, or supercomputing resources. All these resources are generally linked via a middleware that provides the necessary abstraction to enable the exploitation and use of the infrastructure. Some of the state-of-the-art e-Science platforms include grid[2] middlewares such as EGEE gLite, Globus and Unicore [1, 2, 3].

Large scale distributed e-Science infrastructures exist in the form of the Enabling Grids for E-sciencE (EGEE) project , the Open Science Grid, TeraGrid, as well as smaller national-scale grids such as the National Grid Service [4, 5, 6, 7]. Grid infrastructures were initially developed to support "big science" projects, such as Astronomy and Particle Physics. However, due to the transformative nature of these infrastructures e-Science technologies are being used in an ever increasing number of scientific domains. Researchers typically access e-infrastructures in order to use them to run a number of compute-intensive analyses on their data. These analyses take a significant amount of time and have interdependencies that need to be managed. Additionally, due to the distributed nature of e-infrastructures, the execution of these analyses on the distributed resources must also be orchestrated. Workflows are the primary technology that are used for these purposes. In Section 1.1, scientific

[1] John Taylor, Director of Research Councils, Office of Science and Technology, UK

[2] Grids have been considered synonymous with e-Science infrastructures. However recently they are no longer considered the exclusive e-Science infrastructure.

workflows are discussed in detail.

One domain where e-Science is increasingly being applied is the study of *neurodegenerative diseases* [8]. A neurodegenerative disease is one in which neurons are progressively damaged. Examples of neurodegenerative conditions include Alzheimer's and Parkinson's disease. These illnesses mostly occur in elderly people. As the populations of the world are ageing, particularly in Europe, such diseases are becoming more and more common. The problem with such diseases is that thus far they are incurable. In fact, even the diagnosis of these diseases is extremely difficult. Generally speaking, the analysis and diagnosis of these conditions often requires large-scale computing infrastructures. Thus far, research in this domain has been largely hindered by the absence of such platforms. In Section 1.2, one application of e-Science in this domain is discussed.

1.1 Scientific Workflows

Raw e-Science data (for example MRI brain scans, data from a high energy physics detector or metric data from an earth observation project) need to undergo a series of computations before meaningful knowledge can be derived. One way to describe these series of computations on raw e-Science data are workflows. Workflows have emerged as the principal mechanism for describing and enacting complex e-Science analyses on distributed infrastructures such as grids [9]. They provide domain scientists with a systematic, repeatable and reproducible means of conducting scientific analyses. This allows the analysis process to be divided into distinct steps such as design, execution and analysis of results allowing scientists to concentrate on each step in turn. From a computer science viewpoint on the other hand, workflows encapsulate the specification of the series of computations and the data flows required to achieve a specific goal or data product. This data product may be, for example, a cortical thickness measurement of the brain or a simulated climate model.

Since the 1990s a significant amount of research has been carried out in developing workflow management systems. These workflow management systems provide the necessary mechanisms to enable scientists to construct workflows, to enact them and to retrieve workflow output. Workflows have also been used by the business community to automate business logic as well as to orchestrate various business components and key personnel. There are many similarities between business workflows and scientific workflows, however there are some fundamental differences [10].

Due to the demands of state-of-the-art e-Science applications, scientific workflows are increasing in complexity. This complexity is multi-dimensional [11]; scientific workflows are therefore constantly growing in terms of the number of computations and tasks they carry out. This has given rise to a variety of new challenges for researchers developing workflows. One such challenge is that scientists have to sift through increasingly large workflow component repositories in order to construct their desired workflows. A number of issues are involved in choosing appropriate workflow components. There may be many variants of a single component that perform the same computation with only slight differences. Components also often require various parameter settings. Scientists might not have a clear idea of appropriate settings when they are starting their analyses. Additionally, some components may not be compatible with others or may require specific parameters settings to work correctly.

The issues mentioned previously are compounded by the fact that scientists can only learn about

them through trial and error. This renders workflow composition a tedious, repetitive and time-consuming task. It is interesting to note that quite often different scientists may face similar problems when composing workflows. Their experiments may require performing related or similar analyses on their data. By enabling knowledge sharing among different scientists about their successes and failures, some of the complexities may be alleviated. The next section discusses the role of e-Science in neuroscience.

1.2 E-Science in Neuroscience

Neuroimaging is the primary tool used in the study of neurodegenerative diseases. It includes the use of various techniques to image the human brain to study its structure and function. It has been known for some time that neurodegenerative diseases mark the brain with *biomarkers* or *morphological signatures* [12]. By detecting these morphological signatures, researchers can detect the early onset of the neurodegenerative diseases. The detection of these biomarkers has been the primary focus of research in this domain. However, it is not easy to detect such biomarkers in the early stages of the onset of these diseases since only a small percentage of the brain is affected.

Using a medical imaging technique called MRI, detailed brain scans can be acquired. Due to recent advances, these scans have sufficient spatial accuracy and resolution to detect subtle changes (up to 0.5% in the images of the same individual [12]). Therefore, they are the ideal tool for detecting the early onset of the neurodegenerative diseases described earlier. However, in order to reliably identify the onset of these diseases in a particular individual, they must be monitored for the relevant biomarkers over a significant amount of time. Furthermore, with the standardisation of MRI scans, large, centralised neuroimaging repositories containing scans of many patients have emerged. Neuroimaging data repositories include the Alzheimer's Disease Neuroimaging Initiative (ADNI) [13] and the US-NIH MRI Study of Normal Brain Development (NIHPD) [14]. These repositories and various specialised neuroimaging research centres around the world hold thousands of images and cumulatively hold data set sizes of hundreds of terabytes which will ultimately grow to perhaps tens of petabytes in a few years. The emergence of these repositories have opened up avenues for mass analysis of these scans, allowing neuroscientists to conduct research on a larger scale.

In order to extract the biomarkers from an MRI scan, they must be processed using complex algorithms. Recently, toolkits have emerged that provide different algorithms for processing these scans. These include FMRIB Software Library (FSL) [15], Medical Image NetCDF (MINC) [16] and Statistical Parametric Mapping (SPM) [17]. Using these algorithms is extremely compute-intensive and requires extensive processing power, usually not available to neuroscientists. This is where e-Science infrastructures can be utilised. In the following section, one such infrastructure is discussed.

1.2.1 neuGRID

neuGRID is an e-Science infrastructure that was developed to allow neuroscientists to perform mass analyses of MRI scans to detect biomarkers of Alzheimer's Disease. It was funded by the European Commission under the Seventh Framework Programme [8]. The analyses are carried out by executing complex neuroscience workflows on the MRI scans without knowledge of the underlying distributed infrastructure. The infrastructure developed consists of a set of generic middleware ser-

vices that abstract the infrastructure from the services provided to the neuroscientists. The services provided included:

1) Anonymisation Service for stripping patient identifying data from MRI brain scans.
2) Pipeline Service for enacting and orchestrating workflows.
3) Provenance Service for collecting, storing and managing workflow and data provenance[3].
4) Querying Service for querying workflow and provenance repositories in the neuGRID infrastructure.
5) Glueing Service for acting as a middleman between the neuGRID services and the underlying distributed infrastructure, allowing them to be generic and portable.

During neuGRID it was observed that there were only a handful of technically savvy users who had the expertise to design, construct and debug workflows[4]. Most of the scientists simply treated the workflows as black boxes. This meant that most of the users did not have the technical expertise to construct or modify workflows for their individual needs. A mechanism to encourage the transfer of expertise from more advanced scientists to relatively less advanced ones was missing. This was a hinderance to the large scale analysis of brain scans since to modify the workflows to suit their purposes, the scientists had to request the experts. Thus, scientific progress was impeded to some extent. In order to allow the maximum number of neuroscientists to take advantage of the rich features current e-infrastructure provide, a mechanism for knowledge-sharing was required. Novice users (neuroscientists), as well as other expert users, should be able to learn from the experiences of each other. The focus of this thesis shall be to address this problem and devise a method that will enable knowledge-sharing between expert and novice users.

1.3 Research Goals and Aims

Ever since the mid-seventies, researchers have recognised that capturing and sharing knowledge is the key to building large and powerful systems. A significant challenge in the knowledge capture domain is representing the knowledge in a way that enables reuse. One approach to knowledge sharing is by building recommender systems [18]. The aim of such systems is to build user recommendation systems that suggest options to users and help them in choosing. The suggestions are tailored for specific users based on various criteria like historical evidence. The system, using the evidence, attempts to estimate the users' preferences and suggest options the user is likely to choose. Recommender systems are discussed in greater detail in Chapter 2.

The novel contribution of this research will be to design a recommender system that will assist users in composing workflows using a combination of semantics coupled with historical knowledge sharing. For the purposes of this thesis, historical knowledge is defined as the usage patterns that emerge as users develop workflows. Therefore, an understanding of the workflow composition process is necessary. This will also help in determining where the proposed approach can be applied to improve the design process. This research will also investigate how semantics can be used to improve the suggestion generation process to generate increasingly relevant suggestions. Examples of semantics include descriptions of the workflow components' structure and function as well as the

[3]The author was involved in developing and implementing the Provenance Service in this project.

[4]It must be noted that no part of the research presented in this thesis was undertaken as part of neuGRID. The project only served as a motivational tool as well as to provide the initial requirements.

interrelationships between them.

1.4 Research Scope

The aim of this research is to demonstrate how semantics can be combined with historical knowledge to improve the accuracy of suggestions generated by the system. Therefore, the output of this research is envisaged to be a prototype implementation of such a system. However, this implementation will not be a fully-functional product that can be presented to users. The prototype will focus on generating suggestions but user-oriented aspects such as a graphical interface are out of the scope of this research. Therefore, user testing as well as other related activities such as usability analysis are also deemed out of the scope of this research. The prototype implementation will be evaluated using various other qualitative and quantitative methodologies that do not require user intervention.

1.5 Research Hypothesis and Questions

A significant problem in scientific workflow composition is the volume of the repository of workflow components that scientists have to choose from. As described previously, without prior knowledge, it is not easy for scientists to find the appropriate components for their purposes. Additionally, inter-component compatibility is also an issue. It is difficult for scientists to determine which components are compatible without either prior knowledge or specific annotations by other scientists describing the incompatibilities. This thesis will investigate how to improve the suggestions offered by recommender systems that attempt to address these challenges. Therefore, the following hypothesis is formulated:

"Workflow component semantics along with their historical usage patterns can be used to improve the suggestions offered by recommender systems."

In order to thoroughly understand the drawbacks of existing workflow composition systems, a survey must be conducted and ways to improve them must be identified. This includes studying related systems. Therefore, the first research question is:

Question 1. *What are the limitations of current workflow composition and related systems?*

As mentioned before, additional semantics can help in understanding how components function, what, if any, inter-component incompatibilities exist, and whether a component is related to the scientists' purposes or not. Therefore, the second research question is:

Question 2. *To what extent can workflow component semantics be used to improve the suggestions?*

Historical knowledge-sharing can help various scientists learn from the experiences of other scientists. Historical knowledge is encapsulated in the frequent usage patterns of the components. It is assumed that the more frequently a pattern is used, the more scientists consider it useful. Therefore, a mechanism to extract the frequent usage patterns is required. To do so, the following question must be answered:

Question 3. *What approaches are suitable for mining historical usage patterns and how can they be used to improve suggestions?*

This thesis aims to combine semantics and usage patterns to assist users when designing workflows. In order to do so, a hybrid framework is required that leverages both of these sources of information. Thus, an investigation into how that can be achieved is required:

Question 4. *How can workflow component semantics and historical usage patterns be combined to improve the suggestions?*

1.6 Research Methodology

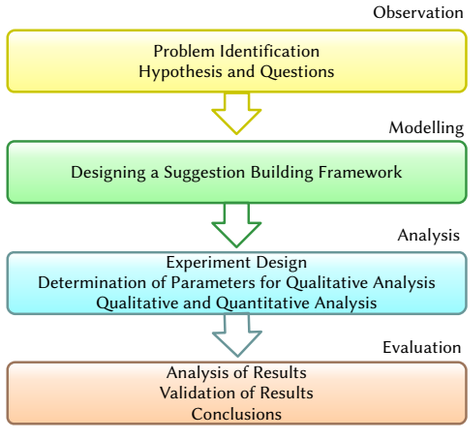


Figure 1.1: Research methodology.

This research shall be conducted in a phased and iterative manner. Each phase of the research is designed to produce a specific outcome that feeds into the next. Figure 1.1 shows all the phases. During the *Observation* phase, the research hypothesis and associated research questions are formulated based on an initial literature review. Additionally, a detailed survey of recommender systems and the workflow composition process will be carried out. Special focus will be given to existing workflow composition tools and their drawbacks. In the *Modelling* phase, a solution to address the drawbacks of existing workflow recommender systems shall be proposed based on historical knowledge sharing. Parameters to perform qualitative evaluation of the framework shall also be determined in this phase. In the *Analysis* phase a prototype implementation of the proposed solution shall be developed. The implementation shall then be evaluated on various datasets using qualitative and quantitative analyses. The results shall then be analysed in the *Evaluation* phase. The results of these will help in proving or disproving the hypothesis and associated research questions. Finally, the research shall be concluded and the findings shall be summarised at the end of the Evaluation phase. The thesis shall be concluded with some suggested future directions.

1.7 Thesis Structure

In Chapter 2, the background and literature survey is discussed. In particular, an overview of recommender systems in general along with their categories and approaches including their application

in workflow systems used is presented. Furthermore, workflow systems in general are also discussed. After the literature review, the suggestion building framework is presented in Chapter 3 along with the various components and what part they play in generating the suggestions. In Chapter 4, the prototype and implementation of the framework along with the challenges faced during the implementation phase are outlined. Chapter 5 presents the evaluation methodology. Chapter 6 presents a detailed account of the evaluation of the framework along with the results and analysis of the results. The thesis concludes in Chapter 7 with a summary of the research outcomes along with their role in proving/disproving the hypothesis. It concludes with suggested future outcomes.

1.8 Publications

K. Soomro, K. Munir, and R. McClatchey, "Incorporating semantics in pattern-based scientific workflow recommender systems: Improving the accuracy of recommendations," in *Science and Information Conference*, London, UK, July 2015, in press.

Bibliography

[1] E. Laure *et al.*, "Middleware for the next generation grid infrastructure," CERN, Tech. Rep. EGEE-PUB-2004-002, 2004.

[2] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, 2005, pp. 2–13.

[3] A. Streit *et al.*, "Unicore 6 — recent and future advancements," *Annals of Telecommunications*, pp. 1–6, 2010.

[4] F. Gagliardi and M.-E. Begin, "EGEE - providing a production quality grid for e-Science," in *Local to Global Data Interoperability - Challenges and Technologies, 2005*, June 2005, pp. 88 – 92.

[5] R. Pordes, "The open science grid–its status and implementation architecture," *JPCS Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP 07)*, 2007.

[6] N. Wilkins-Diehr *et al.*, "TeraGrid Science Gateways and Their Impact on Science," *Computer*, vol. 41, no. 11, pp. 32 –41, November 2008.

[7] G. M. Sinclair, "Trials and tribulations of the UK national grid service," in *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, December 2008, pp. 471 –474.

[8] A. Redolfi *et al.*, "Grid infrastructures for computational neuroscience : the neuGRID example," *Future Neurology*, vol. 6, no. 4, pp. 703–722, November 2009.

[9] E. Deelman *et al.*, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[10] R. Barga and D. Gannon, "Scientific versus business workflows," in *Workflows for e-Science*, I. J. Taylor *et al.*, Eds. Springer London, 2007, pp. 9–16.

[11] Y. Gil *et al.*, "Examining the challenges of scientific workflows," *COMPUTER*, vol. 40, no. 12, pp. 24–35, 2007.

[12] J. Ashburner *et al.*, "Computer-assisted imaging to assess brain structure in healthy and diseased brains," *Lancet Neurol*, vol. 2, pp. 79–88, Feb 2003.

[13] S. G. Mueller *et al.*, "Ways toward an early diagnosis in alzheimer's disease: The Alzheimer's Disease Neuroimaging Initiative (ADNI)," *Alzheimer's & dementia : the journal of the Alzheimer's Association*, vol. 1, no. 1, pp. 55–66, 07 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S155252600500004X?showall=true

[14] "The NIH MRI study of normal brain development (pediatric MRI study)," http://www.pediatricmri.nih.gov/nihpd/info/project_overview.html [Last accessed: 14th Apr, 2012].

[15] S. M. Smith *et al.*, "Advances in functional and structural MR image analysis and implementation as FSL," *NeuroImage*, vol. 23, Supplement 1, no. 0, pp. S208 – S219, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811904003933

[16] R. D. Vincent *et al.*, "Introduction MINC 2.0: A modality independent format for multidimensional medical images," in *Proc. 10th Annual Meeting Organization for Human Brain Mapping*, 2004.

[17] G. Rees, "Statistical parametric mapping," *Practical Neurology*, vol. 4:, pp. 350–355, December 2004.

[18] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 734–749, 2005.

CHAPTER 2

Background and Related Work

Over the years many workflow managements systems have been developed which assist users when designing workflows [1, 2, 3, 4]. They achieve this by checking whether the workflows are syntactically, and in some cases semantically, correct. Moreover, several systems have been developed that enhance the capabilities of these workflow systems and offer proactive suggestions to users while they are designing the workflows. Thus far, there exists a dichotomy in the approach adopted by these systems. They either employ a patterns-based or semantics-based approach. Each approach has its pros and cons that affect how much user effort and expertise is required to use these systems. This chapter shall attempt to explore these capabilities and limitations and pave the way for proposing a solution that combines the two approaches. The aim of this survey shall be to highlight factors that affect the suggestions offered by these systems so that they may be improved.

This chapter introduces and discusses the various concepts and terms that are used throughout this thesis. It presents an overview of the relevant literature along with a critical evaluation. Figure 2.1 shows the roadmap that this chapter follows. The major literature categories along with important papers are shown. The various sections are organised around these categories. The chapter begins by introducing an example neuroimaging workflow in Section 2.1. This is followed by a discussion of the workflow design process in general (Section 2.2). This section highlights the various challenges faced by scientists when designing workflows. Section 2.3 describes various workflow management systems and how they attempt to address these challenges and to what extent. There exist certain intelligent assistants that augment the capabilities of these workflow management systems. These are discussed in Section 2.5 along with their limitations. Section 2.6 generalises the descriptions presented in Section 2.5 and identifies factors that affect the suggestions. Having surveyed these systems, Sections 2.7 to 2.9 cover various techniques they use in the suggestion generation process. Section 2.10 discusses the literature review and Section 2.11 draws conclusions and summarises the chapter.

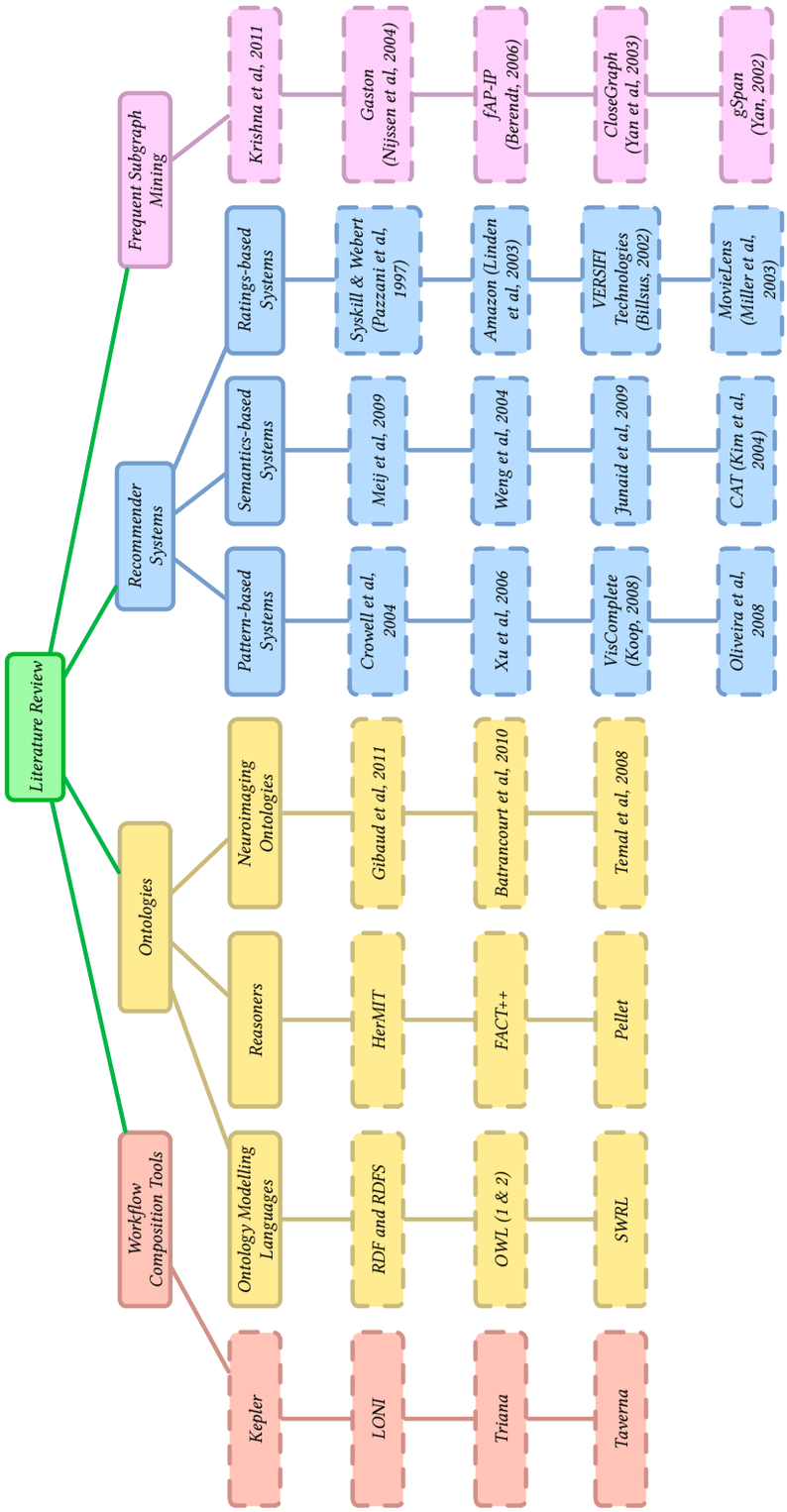


Figure 2.1: Literature review roadmap.

2.1 Case Study

In Chapter 1 the importance of workflows in scientific research was discussed. Neuro-imaging research is one such domain in which workflows are used as an integral part. This section introduces a typical neuroimaging workflow as a case study. This study will serve as a template for examples used in Chapters 3, 4 and 6. The various steps that comprise a neuroscience workflow are summarised below [5]:

1) **Brain extraction**: The MRI scan is stripped of the skull. Brain and non-brain voxels (volume elements) are extracted.
2) **Tissue segmentation**: Voxels representing different types of brain tissues such as grey and white matter, and CSF are separated.
3) **Registration**: Spatial normalisation in which the voxels of an image are matched to a template image. The template may be an earlier scan of the same patient or in case of cross-sectional analyses, a typical brain map.
4) **Statistical comparison**: These determine whether neurodegeneration has occurred. Such deteriorations may indicate the onset of neurodegenerative diseases.

The pivotal step of any neuroimaging workflow is the registration step. This is the step that allows different brains scans to be compared. They may be scans of the same patient taken at different times, or of different patients acquired via different methods (cross-sectional analysis). In addition, errors may arise during the brain extraction and segmentation phases. For example, gradual variations in the intensities of the scanned voxels may occur, resulting in a distortion field known as a bias field. A workflow may also contain steps to correct these errors. Figure 2.2 shows a partial neuroimaging

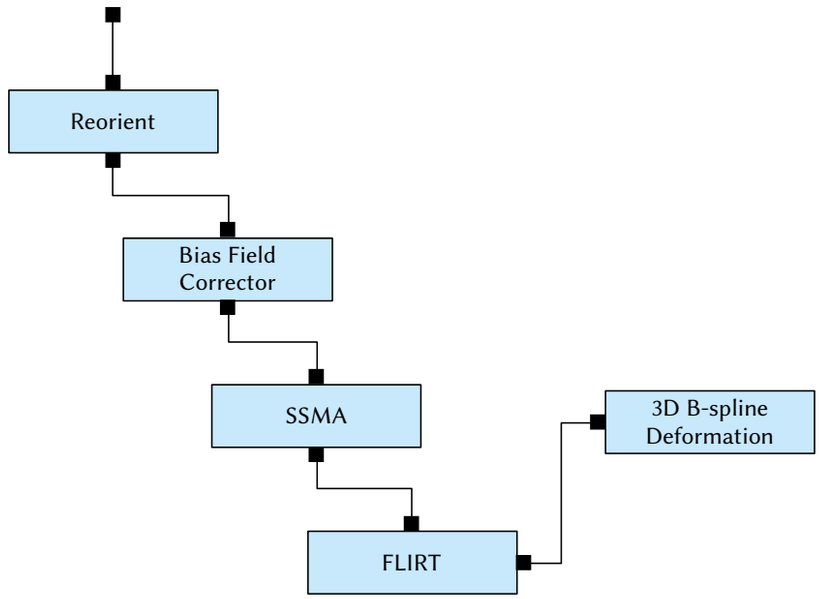


Figure 2.2: A partial neuroimaging workflow.

workflow that extracts 56 regions of interest (ROIs) from the brain and calculates their volumes [6]. The portion shown in the figure takes an MRI scan and rotates it to match the orientation of a reference image. After reorienting the image, bias field correction is applied on it to remove errors from the scanned images. SSMA is an algorithm that performs brain extraction by stripping the skull from the

image, leaving only the brain behind. Once the brain has been acquired, the image is finally registered to the reference image by the FMRIB's Linear Image Registration Tool (FLIRT) algorithm. This step produces a transformation matrix that represents the mapping from the target image to the reference image. 3D B-spline Deformation is an algorithm that takes this transformation matrix and applies it to the target image to align it with the reference image. The rest of the workflow (not shown here) takes the registered image, segments it into various ROIs and calculates the volume of each ROI. These volumes can serve as one statistical measure to determine whether a brain has degenerated or not.

Recall from Chapter 1 that one of the challenges scientists face when designing workflows is the selection of appropriate algorithms. For example, there are several algorithms available for image registration such as FLIRT, FNIRT, Align Linear and Align Warp. Each algorithm has its unique characteristics that make it suitable for use in certain conditions. Each algorithm also has specific requirements for inputs and outputs that distinguish it from other similar algorithms. Without experience, users may not know which algorithms are appropriate for their use case. Similarly, specifying the data-flows and dependencies between these algorithms is also a challenge. Each of these algorithms also take certain input values as parameters. Determining appropriate values for these parameters is another challenge. For example, in Figure 2.2 FLIRT has several input parameters that affect its functioning such as the number of transformation degrees of freedom and the type of interpolation applied. This research aims to design an intelligent assistant that can monitor user actions as they are designing workflows and provide useful suggestions about how to proceed. Therefore, this chapter attempts to identify the capabilities and limitations of existing systems as well as highlight issues that can affect the suggestions. Before existing systems can be discussed, an understanding of the scientific workflow composition process is necessary. The next section attempts to provide this understanding.

2.2 The Scientific Workflow Composition Process

The workflow composition process is depicted in Figure 2.3. It assumes the workflow is being created from scratch. The process begins with a workflow designer looking at a repository of workflow components to identify the appropriate component for their needs (Step 1). Once the appropriate component has been found, the designer adds the component to the workflow (Step 2). With the component in place, the designer then specifies its dependencies and relationships with other components that may already be in the workflow (Step 3). Moreover, the component may require input parameters that affect how it functions. These parameters are also provided at this step. Once the module has been configured, the designer then repeats the process for another component if the workflow is incomplete (Step 1). If the workflow is complete, it is sent to the computing infrastructure for execution (Step 4). The designer then waits for the workflow to execute and the results to be retrieved. The efficacy of the workflow can be determined by looking at the results. If the workflow produces the intended results, the process is complete. If, on the other hand, the results are not SATISFACTORY, then it may be due to several reasons. Either the parameters provided to the various components may not be correct, or the components chosen may not be the right ones. There is no way for the designer to know which of these two cases is true. Therefore, the designer has to adopt a trial and error approach from this point onwards. They must either try different parameters for the components already in the workflow (Step 3), or go back to the repository and try different components (Step 1). This process continues until the

constructed workflow yields the required results.

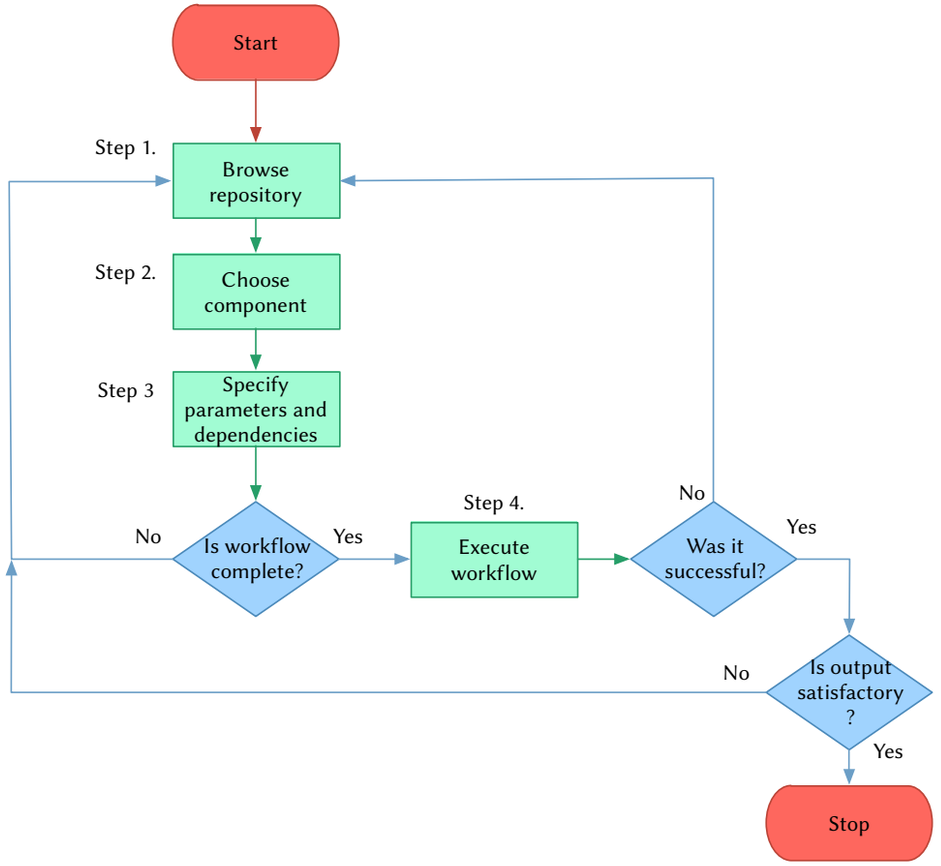


Figure 2.3: The workflow composition process.

To illustrate the aforementioned process, consider the partial workflow shown in Figure 2.2. Suppose that a user wishes to construct this workflow from scratch. There are several starting points the user can choose from. The user can start with the input data, and keep adding components that can process that data until the desired result is achieved. The user can also start by adding the most significant components and then add those required by the significant ones. For example, in the example workflow, the most significant steps required to segment the brain into various ROIs are registration and segmentation. Registration normalises the image according to a template so that different regions in the target image can be reliably identified. Segmentation actually performs the classification of the normalised brain tissue into the required ROIs. As mentioned previously, there are several components that can perform registration. Not all components would be appropriate in this case. For example, FLIRT is a linear registration tool while FNIRT is a non-linear registration tool. Depending on the input image, only one of the two would be appropriate here. Reslicing is another step that is part of the registration process. Again, there are several components like 3D B-spline Deformation, Reslice, Reslice Warp that perform reslicing. However, only some of these components can work together. Therefore, putting these components together requires knowledge, experience and effort. This is one scenario where an intelligent assistant can be helpful for users. In addition, registration cannot be performed on a raw MRI image. It has to be preprocessed to strip the skull and expose the brain. Once again, there are several components, such as BET and SSMA that can perform this operation.

Here again, an intelligent assistant can be helpful. It can determine that the workflow contains a registration step without a brain extraction step and suggest it to the user. Similarly, the assistant can also suggest connections between these components and their configuration parameters.

This entire process is error-prone, time consuming and fragile. Quite often, the repository of components that workflow designers have to sift through is very large. Finding the appropriate component is a tedious task. Each component has some inputs that must be satisfied, and some outputs. Inputs can either be satisfied by connecting to the output of some other component or by providing fixed values that generally remain the same throughout the execution of the workflow. Here again, there is a high risk of error. In the first case, it must be ensured that the output of the previous component is compatible with the input of the latter component. In the second case, determining appropriate values for correct and desirable functioning of the components is often a challenge. It must also be ensured that all the inputs of all the components of a workflow are satisfied and all the outputs of all the components are consumed. This becomes especially difficult when the workflows are large and contain hundreds of components. In addition, workflows cannot have cycles in them (loops in the execution). Cycles make it difficult for distributed infrastructures to determine the order of execution of the components in the workflow. Therefore, workflow management systems generally represent workflows as acyclic graphs. This makes debugging workflows and ensuring correctness a significant challenge. However, ensuring correctness only ensures that the workflow will execute successfully on the computing infrastructure. It may still not produce desirable and useful results. Desirability and usefulness of results can be achieved by using appropriate components, connected properly to each other and provided appropriate parameter settings. Such knowledge comes with experience. Consequently, designing complex workflows is an especially daunting task for novice designers. In the next section, existing workflow management systems and how they address these challenges are discussed.

2.3 Workflow Composition Systems

Recently, workflow composition systems have become an important tool to design and manage scientific workflows. Yu et al [7] present a taxonomy of different workflow systems. There are two main approaches to workflow composition; user-directed and automated. In user-directed systems, users directly edit and compose workflows by hand. In automated systems, users provide descriptions of the kinds of workflows they want along with input and output data descriptions. The system takes these descriptions and generates workflows for the user. Such systems are useful for large workflows containing hundreds of components, but it is very difficult to capture the requirements accurately.

User-directed workflow systems [3, 8] further employ two techniques to modelling workflows; language-based [8, 9] and graph-based [3, 10]. Language-based workflow systems require a user to express the workflow in a workflow representation language. Such systems work reasonably well for simple workflows. However, as the workflows get more complex, the limitations of textual workflow systems become apparent. In graph-based workflow systems [1, 10], users construct graph-like visual representations of workflows. The individual components form the nodes or vertices of the graphs and the connections between them form the edges between the nodes. Graph-based systems make the job of designing workflows considerably easier. However, for large workflows containing hundreds of nodes, visual workflow representations become inconvenient. For this reason, such systems usually

provide some kind of nesting structure for workflows. Using these structures, entire workflows can be incorporated as a single unit in larger workflows. This simplifies the visual representation of the workflow, simplifying the task of designing large workflows for users.

Over the years, many graph-based workflow systems have been developed [11, 12, 13]. They have seen considerable success in domains such as bioinformatics, chemicoinformatics, geoinformatics and neuroinformatics etc. Features provided by these systems include input/output validation, correctness checking and visual construction and editing of workflows. Workflow systems also provide designers with the ability to execute workflows and view results as well as the ability to search the component repository. However, in most cases the validation performed and search capabilities provided by these systems are quite rudimentary. They only notify users of incorrect or invalid workflows, but do not explicitly suggest corrections. For example, LONI Pipeline [10] makes sure a designer does not connect a file output of one workflow component to a parameter input of another component or the output of one to the output of another. Triana [3], being a service-based workflow [14] composition tool, uses WSDL descriptions to describe its components. Therefore, it validates connections by ensuring that two connected components (web services) have compatible XML types as inputs/outputs. Taverna [15] performs some simple type checking that ensures that the inputs and outputs match. For example, a data type can either be a single value or list. Type checking in Taverna involves ensuring that if an output data type is a single value, the input data type should also be a single value. Kepler [16], however, provides complex type checking via ontology-based semantic descriptions. It also allows users to search for components based on these descriptions. LONI and Taverna also provide the capability to add some semantic annotations to the component repository. Designers may then locate components by searching these annotations. However, the annotations are simply free-form text. Therefore, unless a designer knows what they are looking for, it can be difficult to find the right component for their needs, especially if the size of the repository is large. Similarly, Triana also provides similar search capabilities where users have to specify what they are looking for. Kepler is the only system that incorporates formal semantics. This makes it easier for users to search the repository since the semantic annotations are standardised. Goderis et al. have investigated the application of workflow discovery techniques in practice [17]. They conclude that systems that do not take into account semantics when searching workflows do not perform well for rare workflow components. This finding forms the basis of this research and is expounded upon in Section 2.5. The complete features and capabilities of workflow systems are not central to this thesis, therefore, the reader is directed to [11] for a detailed survey of these systems.

Table 2.1 summarises the comparison of the various workflow composition systems. Simple type-checking means that the system only ensures compatible datatypes. No semantic information is incorporated. Complex type-checking includes semantic information as well. Text-based search capability means the system only allows the users to search components using free-form text. On the other hand, semantics-based searching allows users to search components using complex criteria that cannot be captured using free-form text annotations. Finally, as described earlier, in user-directed workflow systems, users construct workflows manually by adding individual components.

For most scientific users, all of the previously highlighted issues are challenges. One way to address them is to suggest components to add to the workflow as designers design them. The workflow can be ensured to be correct by incorporating formal semantic descriptions of the components. The

WMS	Type-Checking	Search Capabilities	Workflow Composition
LONI	Simple	Text-based	User-directed
Triana	Simple	Text-based	User-directed
Taverna	Simple	Text-based	User-directed
Kepler	Complex	Semantics-based	User-directed

Table 2.1: Comparison of various workflow management systems.

issue of finding appropriate components can be addressed by incorporating historical usage patterns along with the semantic descriptions. Historical patterns are defined as the patterns that exist in the way users design workflows. By incorporating formal semantic descriptions and historical usage data, the suggestions can be made increasingly relevant. These approaches are discussed in detail in Section 2.6. Having surveyed existing workflow systems, the following section discusses various intelligent systems that extend the capabilities of these systems by providing intelligent suggestions to users.

2.4 Usability

Usability is an important aspect of any product and as such it has also been given great importance in computer software [18]. The International Organisation for Standardisation (ISO) has developed several standards for usability and how to measure it over time [19]. The latest standard, ISO 25010 specifies usability as consisting of functional suitability, reliability, operability etc among other things [20]. Due to its importance usability measurement and evaluation is an active area of research [21].

Despite the fact that usability is an important aspect of computer software, it has been largely neglected in the case of workflow composition systems. Some systems such as Galaxy have employed standard web usability guidelines to address this challenge [22]. Gordon and Sensen have conducted a pilot study into the usability of Taverna [23]. Similarly Tan et al. have compared the usability of Taverna with the Business Process Execution Language (BPEL) and presented their results [24]. However little has been done to address specific usability challenges in workflow management systems. Usability is an important aspect to consider when designing a system intended to assist users. However, as mentioned in Section 1.4, the purpose of this research is to build a prototype. Therefore, the focus is on the correctness and accuracy of the results only. User-oriented aspects such as usability are out of scope.

2.5 Recommender Systems

Over the past two decades, there has been a lot of interest in developing systems to suggest relevant items to users. In literature, such systems are termed *recommender systems* [25]. Such systems have found many real-world applications such as recommending books, CDs and other products at Amazon.com [26], movies by MovieLens [27] and news at VERSIFI Technologies [28]. Due to the large number of items (objects to be recommended) available, recommender systems help users to simplify the often tedious and cumbersome process of sifting through them. Based on the literature

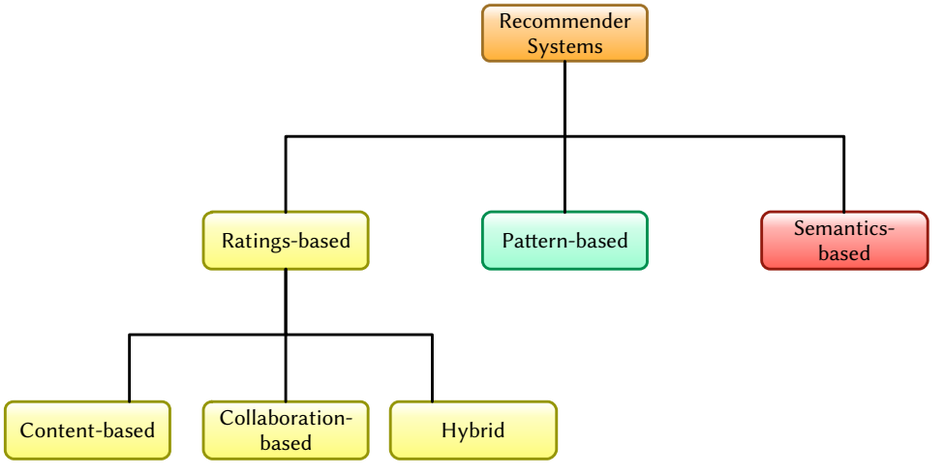


Figure 2.4: A classification of recommender systems.

survey conducted during this phase of the research, a classification model for such systems was developed, shown in Figure 2.4. This model classifies recommender systems according to the strategy they employ to generate suggestions. Each of the categories is discussed in the following sections along with a critical review of their strengths and drawbacks in an attempt to identify factors that affect the suggestions.

2.5.1 Ratings-based systems

Traditionally, the problem of recommending relevant items to users has been most commonly formulated as "estimating ratings for the items that have not been seen by a user" [25, p. 734]. In Figure 2.4, such systems have been classified as ratings-based systems. Quite often, the estimation is based on previous ratings given to the same or similar items. To do so, systems employ either memory-based (heuristics), or model-based (artificial intelligence) techniques. Ratings-based systems are further classified into three categories; content-based systems, collaboration based systems and hybrid systems. Content-based systems try to predict ratings for unseen items based on previous ratings assigned to similar items by the same user. Therefore, content-based systems try to find commonalities between items. Syskill and Webert [29] is one such algorithm that suggests interesting web pages to users based on their topics of interest. Collaborative systems on the other hand find similarities between users. The ratings for a particular user are estimated based on how users with "similar taste" rated those items in the past. GroupLens [30] is an example of a collaborative system for Usenet news. Hybrid systems combine both previous approaches to overcome their individual drawbacks [31, 32].

Ratings-based systems attempt to find items based on similarity. For the problem this thesis attempts to address, the system needs to find items based on necessity. In other words, ratings-based systems try to determine that, given certain items a user has liked in the past, what other items he might be interested in. In contrast, the problem here is that, given certain items that a user possesses, what other items does the user most likely require. Moreover, ratings-based systems require explicit ratings by users. Eliciting explicit ratings from users for every workflow component is not feasible; particularly when the workflow consists of 100s of components. Therefore, this research stipulates that traditional ratings-based systems cannot be used in this case. For this reason, they are only superfi-

cially surveyed here while the focus of this chapter is on pattern-based and semantics-based systems.

2.5.2 Pattern-based Systems

Pattern-based approaches attempt to identify usage patterns using historical data (logs). Frequency of occurrence is an important metric to determine whether a particular pattern exists or not. Pattern-based systems have found application in many domains since the theory behind them is easy to understand and implement. For example, Crowell et al [33] is a pattern-based spelling suggestion system for medical queries. It uses a usage frequency-based ranking algorithm to sort spelling suggestions returned by a spell-checking engine such as GSpell [34] or ASpell [35]. Each of these tools assigns a score to the suggestions it returns based on the degree of matching. The score for each word returned by these tools is combined with a usage frequency extracted from the query logs of a medical database. This combined score is then used to re-sort the list which is then presented to the user. Thus, the suggestions are biased towards more frequently used words as opposed to less frequently used words. The authors stipulate that this bias improves the relevance of the suggestions. Similarly, Xu et al [36] is a recommender system that suggests tags to users for their uploaded content based on tagging patterns. The suggestions are generated when a user starts tagging some content. The system finds other tags that co-occur frequently with the user-supplied tag(s). It also attempts to minimise the number of suggested tags in order to keep the tags specific and relevant while covering as many facets of the content as possible. The converse is also taken into account; the suggested tag combination should not specify a large number of objects. This is done so that users may later be able to find their tagged content with minimum effort. However, workflow composition systems present some unique challenges when compared to such systems. For example, there are explicit links between the different components. In addition, not all components are compatible with each other. Workflow correctness also has to be ensured when constructing the workflow. Any workflow recommender system must take all of these factors into account. Therefore, this thesis focuses on recommender systems for workflow composition only. VisComplete and Oliveira et al are examples of such systems. These systems are summarised below and are critically reviewed at the end.

2.5.2.1 VisComplete

VisComplete [37] is a pattern-based system that treats workflows as graphs and tries to find the most frequent patterns that occur in the graphs. These patterns represent collections of workflow components that designers frequently use in conjunction with each other. When workflows are represented as graphs, frequent subgraphs within that set represent the frequent patterns in the workflows. The suggestions generated consist of entire subgraph structures. However, instead of finding complete subgraphs, VisComplete only attempts to find frequent paths in the graphs. To represent a workflow is a graph, each component in the workflow becomes a vertex in the graph and links between the components become the edges of the graph. Each graph is a Directed Acyclic Graph (DAG).

VisComplete employs a three-step suggestion generation strategy. The first step is to mine the graphs. To extract patterns, the system generates a summary of all paths in the graphs. Because suggestions are generated starting from a particular vertex (called the *completion anchor*), the system extracts all possible paths that begin or end at that vertex. For the second step in the suggestion generation process, the system employs an iterative refinement strategy. For the anchor vertex, it takes all

paths ending or starting at that vertex, and finds all the vertices that are most likely to follow. These new vertices are then chosen as the anchors and for each anchor, the previous step is repeated. To order the suggestions, the system uses a measure of the likelihood of the suggested vertex called the *confidence measure*. The confidence measure is the likelihood of a particular vertex given a particular path. To suggest more than just paths, the system iteratively extends the suggested path at different vertices, thus suggesting trees as well. It stops iteratively refining the suggestions either after a given number of steps or when no new suggestions can be generated. Suggestions are also pruned if their confidence is significantly lower than the parent suggestion's confidence. This difference in confidences is currently implemented as a fixed threshold. Finally, the suggestions are sorted by confidence and presented to the user.

2.5.2.2 Connection Frequency-based Suggestion Generation

Oliveira et al [38] have developed a rudimentary pattern-based recommendation system that they have integrated into the VisTrails [1] workflow composition tool. It works by parsing the repository of workflows and finding frequent connections. For every connection in the repository, it creates a tuple $\langle c_s, p_s, c_d, p_d, wf \rangle$ where c_s is the source component, p_s is the source port, c_d is the destination component, p_d is the destination port and wf is the workflow name. When the list of recommendations for a particular component c is to be generated, the system calculates a recommendation confidence metric for each suggestion. For every connection originating at c, this is the ratio of the number of occurrences of that particular connection in the database to the total number of all connections originating from c. The system only provides single-step suggestions at every point.

2.5.3 Semantics-based Systems

The other category of recommender systems shown in Figure 2.4 is semantics-based systems. These systems use semantic descriptions of the entities to generate suggestions. Meij et al [39] is such a query completion system that uses semantic types of entities to generate suggestions. To achieve this, it divides each query into two parts; an entity and a completion part. The goal of the system is to suggest a relevant completion given a particular entity. To suggest completions, the system looks at the entity type, and picks out the most frequent completions for the entire type rather than the entity itself. This allows the system to suggest relevant completions even if that particular completion does not occur with that particular entity. Weng et al [40] is a system that suggests interesting products to users by extracting a set of features they are interested in. It represents the products as a set of features and determines the level of interest of a particular customer in each feature. It then finds other customers that share interest in the same features. The final recommendations are made on the basis of the purchasing habits of similar customers. Like pattern-based systems, semantics-based approaches also face certain challenges when applied to workflows. Therefore, the focus of this section is on semantics-based workflow composition systems, which are discussed in the following sections.

2.5.3.1 Multi-Criteria Suggestion Generation

Junaid et al [41] have integrated a semantics-based system into the ASKALON workflow environment [2]. It employs a top-down three-step suggestion generation process coupled with user-defined

policies to generate suggestions. When the suggestion generation process is triggered, it uses the

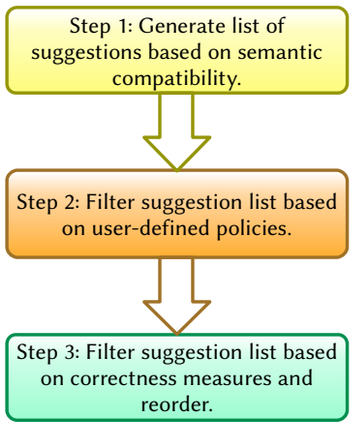


Figure 2.5: Three-step suggestion generation process.

currently selected component as the anchor for the suggestions. Figure 2.5 depicts the suggestion generation process. First it finds all components that are compatible with the anchor. Compatibility is determined by the degree to which the various attributes of the two components match. These include *workflow name*, *workflow domain*, *component function*, *component type*, *component properties*, *component constraints*, *input ports* and *output ports*. A weighted formula is used to calculate a match factor based on these attributes. This value represents a measure of the compatibility of the candidate component with the anchor. Table 2.2 shows the default weights used for these attributes. The user

Attribute	Weight
Workflow name	5
Workflow domain	5
Component type	10
Component function	10
Component constraints	10
Component properties	10
Input ports	25
Output ports	25

Table 2.2: Default weights assigned to various component attributes used during match factor calculation.

can also override these defaults if they so choose. If the match factor satisfies a user-defined threshold, the component is added to the list of suggestion candidates. A high threshold value yields a more precise and small set of suggestions, while a low threshold value yields a larger set of suggestions.

Once the list of the suggestion candidates has been built, the candidates are filtered based on various user-defined criteria. This constitutes the second step of the suggestion generation process. The filtration policy can be used to specify (a) user-specific (b) domain-specific, or (c) time interval-specific filtration rules. For example, the policy can specify that all suggestions from users A and B be ignored or that only suggestions from a specific domain or time period be allowed. In addition, users can also specify any combination of these rules. After the filtration policy has been applied, the suggestions

are reordered and further filtered in the third step. The filtering at this stage is performed on the basis of two measures; the design-time and the runtime reliability and correctness of the suggestions. Design-time reliability and correctness are determined by:

1) The skill level of the users designing the workflows.
2) The frequency of use by expert users.

In order for this to work, the users are classified as novice, intermediate or expert by the system and actions originating from expert users get higher weightage. In addition, suggestions are ranked according to their frequency of use. Runtime reliability and correctness are measured by:

1) The number of successful executions of the component.
2) The degree of correctness of the results.
3) The resources previously used by the component currently available in the grid.
4) The resources previously used by the component currently reserved in the grid.

The correctness of past results is rated either by users or an automated evaluation tool on a scale of one to five. For information about grid resources, the system contacts a grid resource manager. If the correctness weight does not satisfy the user-defined threshold described previously, the suggestion is discarded. Otherwise, this correctness weight is added to the match factor calculated in the first step. Finally, the suggestion list is reordered based on the combined weight of each component calculated in the three steps and the list is presented to the user.

2.5.3.2 Composition Analysis Tool (CAT)

Composition Analysis Tool (CAT) [42] is another example of a semantics-based system. It was developed for the Southern California Earthquake Center (SCEC) and is integrated into their Community Modelling Environment (CME) [43]. CAT represents workflows as a set of components, links that connect these components, initial inputs and end results. It employs a mixed-initiative approach to workflow composition; the system can generate complete or partial workflows automatically from user-defined descriptions as well suggest actions to users as they compose workflows. At every step, the system uses semantic descriptions coupled with formally defined properties to determine correctness of workflows. These include:

1) **Tasked**: a workflow contains one or more end results.
2) **Satisfied**: all input parameters of all components are provided by other components, by default values, or as user inputs.
3) **Grounded**: all components are executable and not abstract. These are explained in the subsequent discussion.
4) **Justified**: at least one output parameter of each component is linked to an end result or the input parameter of a Justified component.
5) **Consistent**: each link in the workflow connects the output parameter of one component to the input parameter of another component, where the latter subsumes the former.
6) **Unique**: no link or component is redundant.

The semantics of the components and their input and output ports are described in a knowledge base using two separate hierarchical ontologies; a component ontology and a domain term ontology. The component ontology specifies the relationships between the various components and their descriptions along with default parameter values. The domain term ontology specifies the relationships be-

tween the data types of the input and output ports of the components. It must be noted that not all components in the component ontology are executable. Some components are abstract and represent the most common features of a set of components. As mentioned previously, if a workflow contains an abstract component, it violates the Grounded property. The abstract component must be specialised to an executable one for the workflow to be correct.

To generate workflows automatically, users provide descriptions of either the desired results, the kind of workflow they want or of the initial data. The system uses AI planning techniques to generate a partial or complete workflow that satisfies both the semantic constraints as well as all the correctness properties. If more than one component can be used at any point, the system uses an abstract component as described by the component ontology. Once a workflow is returned to the user, he now has the option to modify it. On the other hand, if the user wishes to construct a workflow manually or modify an existing workflow, the system critiques the user's actions at all times and ensures that the workflow remains correct. If the workflow does not satisfy any of the previously mentioned properties, CAT suggests a remedial action to the user. In this manner, the system tracks the user's actions and provides suggestions to him.

2.5.3.3 CBR-based Systems

In recent years semantics-based systems have employed Case-Based Reasoning (CBR) techniques for suggestion generation [44, 45]. Such systems treat the repository of workflows as the so-called case base used in CBR systems. When workflows are being constructed semantic descriptions of the inputs and outputs of components are compared with the workflows in the case base and similar workflows or subworkflows are retrieved. Having surveyed recommender systems, the following section identifies factors that affect the suggestions generated by these systems. Identifying these factors will help in assessing the efficacy of these systems and pave the way for improving them.

2.5.3.4 Assisted Service Composition Systems

When talking about semantics-based recommender systems, assisted service composition systems are also relevant. These systems assist users in composing services using pre-defined tasks. This is analogous to composing workflows using pre-defined workflow components. As with semantics-based workflow recommender systems, assisted service composition systems leverage the semantics of available tasks such as their inputs and outputs to suggest next steps to users. Several such systems have been developed [46, 47]. On the other hand some systems focus on automatic service generation based on user constraints instead of assisting them at each step [48, 49]. Again such systems are analogous to automatic workflow generation systems [12]. Semantics-based assisted service composition systems suffer from the same drawbacks as their counterpart assisted workflow composition systems (cp. Section 2.10).

2.6 Factors Affecting Suggestions

Since the main job of recommender systems is to provide suggestions to users, the efficacy of these systems depends on the suggestions presented. It can be measured via the rank of the required suggestion in the list. The closer the suggestion is to the top, the better the performance; the lower the

suggestion in the list, the lower the performance. Generalising the survey presented in Sections 2.5.2 and 2.5.3, it can be argued that the suggestions depend on several factors. These are discussed in the following sections along with a summary of their state-of-the-art in existing systems.

2.6.1 Frequent Usage Patterns

Frequent usage patterns help to identify which components are most appropriate for a particular purpose. Scientific users in particular are generally not adept at dealing with computer-related issues. It was observed during neuGRID [50] that due to the issues highlighted in Section 2.2, only a few users take the time and effort necessary to familiarise themselves with the workflow composition process. Due to this, most users only act as workflow consumers and have to rely on a handful of expert users to construct and modify workflows for their needs. By leveraging the frequent usage patterns of other users during suggestion generation, novice users can take part in the workflow design process as well. In addition, some components may not be compatible with each other. As users use the system, such components will appear only rarely in the system since they will not generate workflows that provide correct results. Therefore, the suggestions will be automatically biased towards compatible components. Incorporating frequent usage patterns also makes the system dynamic. It allows the system to learn from user actions as it is used. Therefore, as users acquire expertise and learn better ways to achieve their desired results, the system can learn along with them. All of these factors would improve the system.

However, it may also be the case that rare components are more appropriate for a user's needs than frequently used ones. This is especially true if a new component is added to the repository. Since it is a new component, there will not exist many patterns that have that component in them. Using frequent patterns for suggestion generation has the drawback of ignoring such components. Both VisComplete and Oliveira et al suffer from this drawback. Another drawback of using frequent patterns is that when the system is new and patterns do not yet exist, the system will not be able to assist users. These factors would decrease the efficacy of the suggestions.

2.6.2 Relevance of Results

For any suggestion system, the ultimate goal is to assist the user in achieving their final result. For a spelling suggestion system, the user must get the word they are looking for. For a search engine, it is important that users get all the relevant results that they require. Similarly, for a workflow composition system, it is important that the final workflow yield results that the users require. Therefore, if a particular workflow provides the required results, giving it a higher weight would bias the results towards successful workflows. Without this, all workflows would have equal weighting. However, not all workflows provide results with the same level of relevance. Therefore, biasing the suggestions towards more relevant workflows would improve them. Currently only Junaid et al incorporate relevance of results in their suggestion generation strategy. This also affects the suggestions.

2.6.3 Expertise of Users

When novice users construct workflows, they generally do not have a very good idea of the kind of components they want. Because of this, they tend to make mistakes including constructing irrelevant

workflows. Expert users, on the other hand, are much less likely to make such mistakes. They are more likely to construct useful and relevant workflows. Without discriminating between expert and novice users, the system would treat patterns originating from both equally. This would affect the relevance of the suggestions. Irrelevant workflows may be considered noise in the data. Since they are likely to originate from novice users, by giving them a lower weight, their effect can be tempered. This, coupled with the relevance of the final results might provide sufficient information to generate relevant suggestions. Therefore, by incorporating the expertise of the users during pattern extraction, another positive bias in the suggestions can be introduced. At the moment only Junaid et al considers this bias in their suggestions.

2.6.4 Semantic Metadata of the Workflow Components

Semantics about the workflow components can be used by the system to reason about them and determine which components can be connected together. In addition, the system can determine if two components that are semantically incompatible can be made compatible by interposing more components. CAT adopts this approach, but it is only limited to adding one component. The system does not check if adding multiple components can help. Semantics about the workflow components can help the system find such components to suggest to the user. In addition to this, if the components are described using a hierarchy of types in the knowledge base, then users can add generic component types to the workflow. The system can then suggest specific appropriate components of that type to the user. CAT uses a similar approach. Moreover, as discussed in the previous section, in rare cases infrequent components may be more suitable for particular users needs than frequent ones.

The drawback to this is that the user needs to know that a rare component exists that is more suitable. Therefore, semantics-based suggestions are mostly useful for expert users. Another drawback to semantics-based approaches is that they require a one-time concentrated effort by users to define the semantics of the components. Without this effort, the system is unable to generate suggestions. The system is also static. Once the domain knowledge has been encoded into the knowledge base, the system does not automatically add new knowledge to it. Therefore, as users become expert and learn new patterns, the system does not evolve along with them. The knowledge base has to be manually updated by the users. The system can be made dynamic by incorporating frequent usage patterns. Another important factor in workflow design is parameters. Their importance is discussed in the next section.

2.6.5 Parameters

Parameter values are very important for the correct function of workflows. Inappropriate parameter values can cause components to function incorrectly, causing the results to be inaccurate or incorrect altogether. Determining appropriate parameter values is a non-trivial task in most cases. Usually, the optimum value of a parameter is determined through parameter sweep workflows [51]. For this type of analysis, the same workflow is executed with a range of different values for its parameters. This method is inefficient since each workflow instance may take a long time to execute. The total time it takes to find the optimum parameter value increases exponentially with this method. Once the optimum value of a parameter is learned, then it must be shared with other users. CAT allows this by allowing users to define a default value for different parameters in the knowledge base.

It then suggests that value to users. However, quite often parameter values are dependent on the specific type of data that is provided as input. Therefore, it would be much more useful to provide parameter value suggestions based on the context. An intelligent system can learn this by extracting the most frequently used values in a specific context. Then, when the user constructs a workflow in that particular context, the system can suggest the frequent parameter values.

2.6.6 Ranking Method

When the suggestions are presented to the user, the ranking method becomes particularly important. The better the ranking method, the more relevant the results. Traditionally the ranking method is based on the frequency of use. This ensures that the most frequently suggestions are presented at the top. However, in the situation where a rare component is what the user needs, this method would either rank the component very low, or not suggest it at all. One solution to this is to have alternate ranking systems that the users can switch between at will. CAT uses a heuristics-based ranking method. The most recent suggestions are ranked at the top, while those generated further back in time are ranked lower. Without any bias in the ranking of the suggestions, there is no guarantee the system will suggest the most relevant components.

2.6.7 Workflow Correctness

It is important for a workflow system to be able to tell users that the workflows they are designing are correct. There are two aspects to correctness; structural correctness and semantic correctness. Structural correctness insures that all components are connected properly and that there are no cycles in the workflow. Semantic correctness ensures that all the components connected to each other are semantically compatible; the data types of their inputs and outputs match. Workflow composition systems such as Taverna and Triana only perform structural correctness checking and rudimentary semantic checking. They inform users if a workflow is not correct, but they do not actively offer suggestions to remedy it. A few systems like CAT and Kepler perform complex semantic correctness checking. CAT also uses the correctness checks to determine when components need to be removed and when some components are missing. This feature improves the suggestions. The reader is directed to Section 2.3 for an overview of workflow composition systems.

2.6.8 Depth of Suggestions

The depth of the suggestions means the number of components the system suggests at a time to users. If the system only suggests one component at a time, then the user has to perform at least one click to select each component. This makes the user's job more difficult, especially when the workflows the users are constructing are quite large (100s of components). Therefore, suggestions of greater depth may be better. CAT handles this by providing automatic generation of workflows; users describe the kind of workflows they need including inputs and outputs, and the system generates a workflow automatically. The user can then modify the workflow as they require. However, it is often difficult to capture user requirements succinctly and completely. VisComplete on the other hand handles this by suggesting multiple components at one time. As opposed to CAT, VisComplete's approach bases its suggestions on frequent patterns, since it is difficult to tell what users want without having them

provide a description of their workflows.

2.6.9 Comparison of Recommender Systems

Table 2.3 presents a comparison of the various workflow recommender systems discussed earlier. Semantics-based systems such Junaid et al and CAT require significant user effort to start generating

System	Effort	Parameters (P)	Strategy (S)	Depth (D)	Ranking (R)	Correctness	Relevance	User Expertise
Junaid et al.	Concentrated effort required	×	S_s	D_s	R_f	×	✓	✓
CAT	Concentrated effort required	✓	S_s	D_s	R_h	✓	×	×
VisComplete	Little effort required	×	S_p	D_m	R_f	×	×	×
Oliveira et al.	Little effort required	×	S_p	D_s	R_f	×	×	×
CBR-based sytems	Concentrated effort required	×	S_s	D_m	R_h	✓	×	×

Effort: How much effort is required by users before system can start generating suggestions.

Parameters: Does the system provide parameter value suggestions?; ✓ = Yes, × = No.

Strategy: What strategy does the system employ to generate suggestions?; S_s = Semantics-based strategy, S_p = Pattern-based strategy.

Depth: Number of components suggested at a time; D_s = Single action, D_m = Multiple actions.

Ranking: Ranking strategy for suggestions; R_f = Frequency-based, R_h = Heuristics-based.

Correctness: Does the system incorporate correctness checking for workflows?

Relevance: Does the system incorporate relevance of the results?; ✓ = Yes, × = No.

User Expertise: Does the system consider the expertise of the users?; ✓ = Yes, × = No.

Table 2.3: Recommender systems and their features.

suggestions. The reasons for this have already been discussed in Section 2.6.4. On the other hand, pattern-based systems do not require much user effort (Section 2.6.1). Generally, all systems provide only single components as suggestions except for VisComplete. Most components rank the suggestions by frequency. Only CAT employs heuristics to rank the suggestions. In addition, CAT is the only system that incorporates formal workflow correctness checking. Junaid et al is the only system that incorporates relevance of the final results as well as user expertise. This thesis will investigate how all these factors affect the suggestions. Assisted service composition systems have been excluded from this comparison since they share their pros and cons with semantics-based systems. Having surveyed various workflow recommender systems, a discussion of the technologies they use to represent the knowledge and patterns is presented. The recommenders use these techniques to generate suggestions.

2.7 Ontologies

An ontology is defined as "an explicit specification of a conceptuali[s]ation" [52]. A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. In this case, the purpose is to provide context to data or entities for intelligent reasoning. Ontologies provide semantic information about real world objects, while actual descriptions of the objects are called ontology instances. Usually, they are modelled using different modelling languages. The languages

themselves rely on XML-based syntax. Therefore, XML acts as a meta-language in which all ontology expression languages are written. These are discussed in the following sections.

2.7.1 RDF & RDFS

Resource Description Framework (RDF) is a model used to represent information about objects (resources). A resource description in RDF is a 3-tuple (triplet) that consists of a *resource*, a *property* that is to be specified for that resource, and a *value* for that property. RDF itself provides a domain-neutral mechanism to describe individual objects. Each triplet is nothing more than a statement describing some aspect of a resource. It does not define the semantics of any application domain, nor specific assumptions about a particular domain. In other words, RDF is better suited to specify instances of ontologies, not the ontologies themselves. Ontologies are better encoded in RDFS.

Unlike RDF, RDF Schema (RDFS) provides a vocabulary to encode domain concepts like classes and specify relationships between them. Using these features, ontology designers can also create domain taxonomies. In other words, RDFS provides the domain-specific semantics about entities that RDF does not capture. For example, the statement that dog is an animal and has legs is a domain concept. To express such concepts, the RDFS vocabulary includes terms such as *Class*, *subClassOf*, *Property*, and *subPropertyOf*. The aforementioned statement can be expressed in an RDFS vocabulary by creating three classes called dog, animal and leg. A property called has-leg can be defined that links the class dog to the class leg while the classes dog and animal can be linked via a subClassOf property. It must be noted that while RDFS is better suited to asserting general statements about concepts, RDF is better suited to assert statements about instances of those concepts. For example, the statement "Tom is a dog" can be encoded as an RDF triple (Tom, rdf:type, dog). Combining this with the RDFS semantics, it can be inferred that Tom is an animal with legs. Even though RDFS provides useful constructs for expressing ontologies, it is quite simple compared to other full-fledged knowledge representation languages. These are discussed in the following sections.

2.7.2 DARPA Markup Language+Ontology Inference Layer (DAML+OIL)

DAML+OIL extends RDFS by allowing further constraints and relationships among objects and their properties to be specified. These include cardinality, domain and range restrictions, and union, inverse and transitive rules. Its syntax for classes and instances is the same as RDFS. DAML+OIL distinguishes between the types of entities that exist in an ontology; those that represent objects that are members of the various classes and those that represent XML datatypes. Similarly, properties are also of two types; those that relate objects to other objects and those that relate objects to datatype values. DAML+OIL is much more expressive than RDFS in terms of representing domain knowledge. For example, using plain RDFS, one cannot specify that a dog can only have four legs. However, DAML+OIL provides the capability to specify cardinality restrictions. Therefore, one can say that a dog is related to exactly four instance of class leg via the has-leg property.

2.7.3 Web Ontology Language (OWL)

Web Ontology Language (OWL) is a successor to DAML+OIL. Its vocabulary includes a set of well-defined XML elements and attributes that are used to describe domain terms and their relationships

in an ontology. OWL's vocabulary is extremely rich for describing relations among classes, properties and individuals. For example, relationships such as *Symmetric*, *InverseOf*, *equivalentProperty* and *Transitive* can be defined. There can be a specific *cardinality*, *minCardinality* or *maxCardinality*. A class can be an *intersectionOf*, *unionOf* or *complementOf* another class. These are all examples of the richness of the OWL vocabulary. For example, consider an ontology of familial relations. If A is a sibling of B, that implies that B is also a sibling of A. Such two-way properties are called symmetric properties. Previous ontology modelling languages did not have the expressiveness to capture this relationship. However, OWL allows one to do so by specifying the has-sibling relationship as symmetric.

OWL is a combination of three increasingly expressive sublanguages; *OWL Lite*, *OWL-DL* and *OWL Full*. OWL Lite is intended to support the building of simple classification hierarchies with simple constraints. For example, cardinality can only be 1 or 0 in OWL Lite. OWL-DL uses description-logics [53]. It provides maximum expressiveness but also ensures that all conclusions are computable and will finish in finite time. OWL Full supports maximum expressiveness at the expense of computational completeness and decidability. It can be viewed as an extension of RDF, while OWL-DL and OWL Lite can be viewed as extensions of a restricted view of RDF.

2.7.4 Web Ontology Language (OWL) 2

OWL 2 extends OWL 1 by adding further expressiveness. The new features are divided into two categories; those that were possible to express in OWL 1 and those that weren't. New terms have been added to the vocabulary for features that could already be expressed in OWL 1 such as *disjointUnionOf*, *negativeObjectPropertyAssertion* and *negativeDataPropertyAssertion*. New features that could not be expressed in OWL 1 include self restrictions, qualified cardinality restrictions, reflexive, irreflexive and asymmetric properties, and property chains. For example, in a familial relations ontology, two individuals cannot be both siblings and parent-child at the same time. Therefore, these two relationships are mutually exclusive. There was no way to capture these kinds of semantics in OWL 1. However, this has been made possible in OWL 2 through the use of disjoint properties. One can now specify that the properties has-sibling and has-father and has-sibling and has-child are pairwise disjoint.

2.7.5 Semantic Web Rule Language (SWRL)

SWRL is a rule-based semantic language that extends OWL Lite and OWL-DL. It adds inference capabilities to OWL that are difficult to achieve otherwise. A SWRL rule is a combination of an antecedent and a consequent. The antecedent is a set of conditions that if true, imply that the consequent must also be true. For example, if two individuals share the same parents, then it must follow that they are siblings. Such a rule cannot be directly encoded in OWL. However, doing so in SWRL is rather trivial.

2.7.6 Comparison of Ontology Modelling Languages

This section presents a comparison of the various ontology modelling languages discussed previously. Since RDF cannot specify domain semantics, it is better suited to instantiating ontologies than modelling them. For this reason, it is excluded from this comparison. On the other hand SWRL can model complex semantics, but those models are still expressed using OWL. Consequently, it is also

excluded from this comparison. Therefore, only RDFS, DAML+OIL, OWL 1 and OWL 2 are compared in Table 2.4. The comparison covers the major features that affect the expressiveness of the respective languages. Some of these features, such as Datatypes and Collections exist in all languages, albeit with varying degrees of utility. As can be seen, there isn't much of a difference between the expressiveness of DAML+OIL and OWL. In fact, the major difference between the two is that OWL employs a better syntax. OWL 2, however, is a significant improvement over OWL 1.

Feature	RDFS	DAML+OIL	OWL 1	OWL 2
Class and Property Subsumption	✓	✓	✓	✓
Domains and Ranges	✓	✓	✓	✓
Datatypes	✓	✓	✓	✓
Collections	✓	✓	✓	✓
Cardinality Restrictions	x	✓	✓	✓
Qualified Cardinality Restrictions	x	✓	x	✓
Complementary Classes	x	✓	✓	✓
Property Value Restrictions	x	✓	✓	✓
Disjoint Classes	x	✓	✓	✓
Equivalence	x	✓	✓	✓
Enumerations	x	✓	✓	✓
Transitive Properties	x	✓	✓	✓
Symmetric Properties	x	✓	✓	✓
Keys	x	x	x	✓
Property Chains	x	x	x	✓
Asymmetric, Reflexive and Disjoint Properties	x	x	x	✓
Class Punning	x	x	x	✓

Table 2.4: Comparison of various ontology modelling languages.

2.7.7 Neuroimaging Ontologies

The use of semantics in neuroimaging is a fairly recent phenomenon. OntoNeuroLOG [54] is an application ontology developed as part of the NeuroLOG project. It extends the OntoNeuroBase [55] ontology developed for NeuroBase. The NeuroLOG project is designed to provide semantics-based querying capabilities to users. To that end they have developed an application ontology to classify various kinds of operations that can be performed on neuroimaging workflows as well as the datasets produced and consumed by them. Moreover, they have also designed an ontology of the methods through which neuroscientists collect data about different subjects [56]. OntoNeuroLOG has been developed using the OntoSpec methodology [57]. Using this methodology, ontology designers are required to justify various design decisions so that any logical or semantic inaccuracies can be identified during the design phase. The next section discusses various methods for reasoning over ontologies.

2.8 Semantic Reasoners

Ontologies themselves are static models of domain semantics. This model is called the asserted model. To be useful in most real-world applications, some kind of reasoning must be performed on them. This reasoning capability adds dynamism to the semantic model. It is essentially what provides intelligence to an artificial system. The model that is obtained as a result of this reasoning is called the inferred model. To obtain the inferred model, a number of semantic reasoners have been created [58, 59, 60, 61]. Reasoning tasks can be distinguished into two types; Tbox and Abox [62]. Tbox reasoning tasks are those related to the structure of the ontology; the classes, their properties and the relationships between them. Checking whether the various classes in an ontology can have instances or not and whether one class subsumes another class are typical examples of Tbox reasoning. Abox reasoning tasks, on the other hand, relate to the instances of the various classes that make up the ontology. These include retrieving all the instances of a particular class and getting property fillers for different instances. Therefore, the capability to perform Tbox and Abox querying provides one criterion for evaluating reasoners.

In addition to simple reasoning tasks, real world applications often require complex querying capabilities that allow applications to retrieve parts of the ontology model based on certain criteria. Conjunctive queries are a mechanism to achieve this feature. Such queries can be expressed in languages such as SPARQL [63] and SPARQL-DL [64]. Typically, conjunctive queries are executed on the asserted model. However, for most complex real world applications, they must be performed on the inferred model rather than just the asserted model. In order to achieve this, some reasoners support conjunctive Abox queries. These queries are a conjunction of statements that comprise the query criteria. Therefore, support for conjunctive queries provide the second criteria for evaluation.

Another mechanism for adding dynamism to ontologies are semantic rules. They can be expressed in languages such as SWRL (cp. Section 2.7.5). Rules are particularly useful when new relationships and new classifications are to be inferred about the instances in the ontology. To be able to do so, the reasoners must have support for rules. Therefore, they become another criteria for evaluating them. Table 2.5 shows various reasoners compared according to the criteria discussed previously. Only OWL reasoners have been chosen for the purposes of this discussion since that is the language that has been chosen for expressing the semantics required for this research. Reasons for this are discussed in Section 2.10. The next technique recommender systems use to generate suggestions is frequent patterns. The problem of finding frequent patterns can be formulated as finding frequent subgraphs in a set of graphs. The reasons for this are also discussed in the next section.

Feature	Types of Reasoning	Conjunctive Queries	Rules
Pellet	Tbox, Abox	SPARQL, SPARQL-DL	✓
RacerPro	Tbox, Abox	SPARQL	✓
HermiT	Tbox	-	✓
FaCT++	Tbox, Abox	-	-

Table 2.5: Comparison of various semantic reasoners.

2.9 Frequent Subgraph Mining

As discussed in Section 2.5.2, the problem of extracting frequent patterns from workflows is often formulated as extracting frequent subgraphs from a set of graphs. This is an intuitive approach because workflows lend themselves to being modelled as graphs because of their inherent graph-like structure. Frequent subgraph mining is a well-researched area and many efficient algorithms exist [65]. For this reason, this research uses existing algorithms instead of implementing a custom one. One of the main challenges in frequent subgraph mining is how to represent a graph internally so that different graphs can be efficiently stored and compared. Detecting isomorphic graphs is also a challenge since a graph mining algorithm aims to output each frequent subgraph only once. To detect isomorphism, a canonical representation is used for the graphs. It is defined as a representation that is the same for all isomorphic graphs. Another challenge is the search strategy employed, e.g. depth first search or breadth first search. All of these factors affect the memory requirements and computational time of the algorithms. In addition, there are other factors that affect the choice of algorithm such as completeness of output and nature of input. Figure 2.6 shows various frequent subgraph mining algorithms.

Figure 2.6: Classification of various graph mining algorithms [65].

For the purposes of this research, an algorithm that takes several graphs as input and finds all possible frequent subgraphs is required. Based on these criteria, the following ones have been shortlisted:

- FARMER
- gSpan
- FFSM
- Gaston

Krishna et al [65] have benchmarked various algorithms and compared their performance. Their benchmarks show that Gaston [66] and gSpan [67] perform much better than other algorithms for various input datasets. Moreover, C and Java implementations of Gaston and gSpan are readily available

Algorithm	Language	Runtime (secs)
Gaston	C	0.5
	Java	4.9
gSpan	Java	3.008
	C	Crashes for large outputs

Table 2.6: Comparative runtimes of different frequent subgraph mining algorithms.

online. Therefore, these algorithms were chosen for evaluation. For these evaluations, a dataset of 73 neuroscience workflows was chosen from the LONI workflow repository. These workflows were first converted into a set of graphs. The various components become nodes in the graphs and links between them become edges. The graphs were then passed through a frequent subgraph mining algorithm. For these tests, a minimum frequency threshold of 4 was chosen. Figure 2.7 shows the results of these tests. The graphs depict the number of subgraphs of various sizes found for each frequency greater than 4. As can be seen, the different algorithms found approximately the same number of subgraphs. Table 2.6 shows the execution times of the various algorithms. The C implementation of Gaston is the fastest while the Java implementation of gSpan performs better than the Java implementation of Gaston. The C implementation of gSpan crashes if the output dataset is too large.

2.10 Discussion

Based on the literature review, the following shortcomings have been identified in existing workflow recommender systems:

1) Pattern-based systems [37, 38] cannot generate suggestions unless sufficient data is available for them to find useful patterns. In addition, since pattern-based systems prune suggestions and only keep the most frequent ones, rare cases where the less frequently used components are more appropriate for the user's needs get ignored. Therefore, in this scenario, the suggestions would be both less relevant as well as misleading for the user.
2) Pattern-based systems do not provide parameter suggestions and only focus on links and components. However, parameters are an important factor for workflows to function properly and they should be included as an important aspect of pattern extraction. Both VisComplete and Oliveira et al suffer from these drawbacks.
3) VisComplete extracts all paths from the workflow graph and uses them to generate suggestions. However, since the suggested graph structures may not be just paths, the system has to use an iterative refinement strategy coupled with heuristics to choose vertices to expand from to suggest trees. This increases the computation required at suggestion generation time. If, instead of just paths, trees are also extracted during the pattern extraction phase, some of the computation required at suggestion generation time can be reduced. On the other hand Oliveira et al only extracts individual links between components. The drawback of this is that the system only suggests one component at a time. Instead, users may find it more useful to have multiple components suggested in one go such as VisComplete.

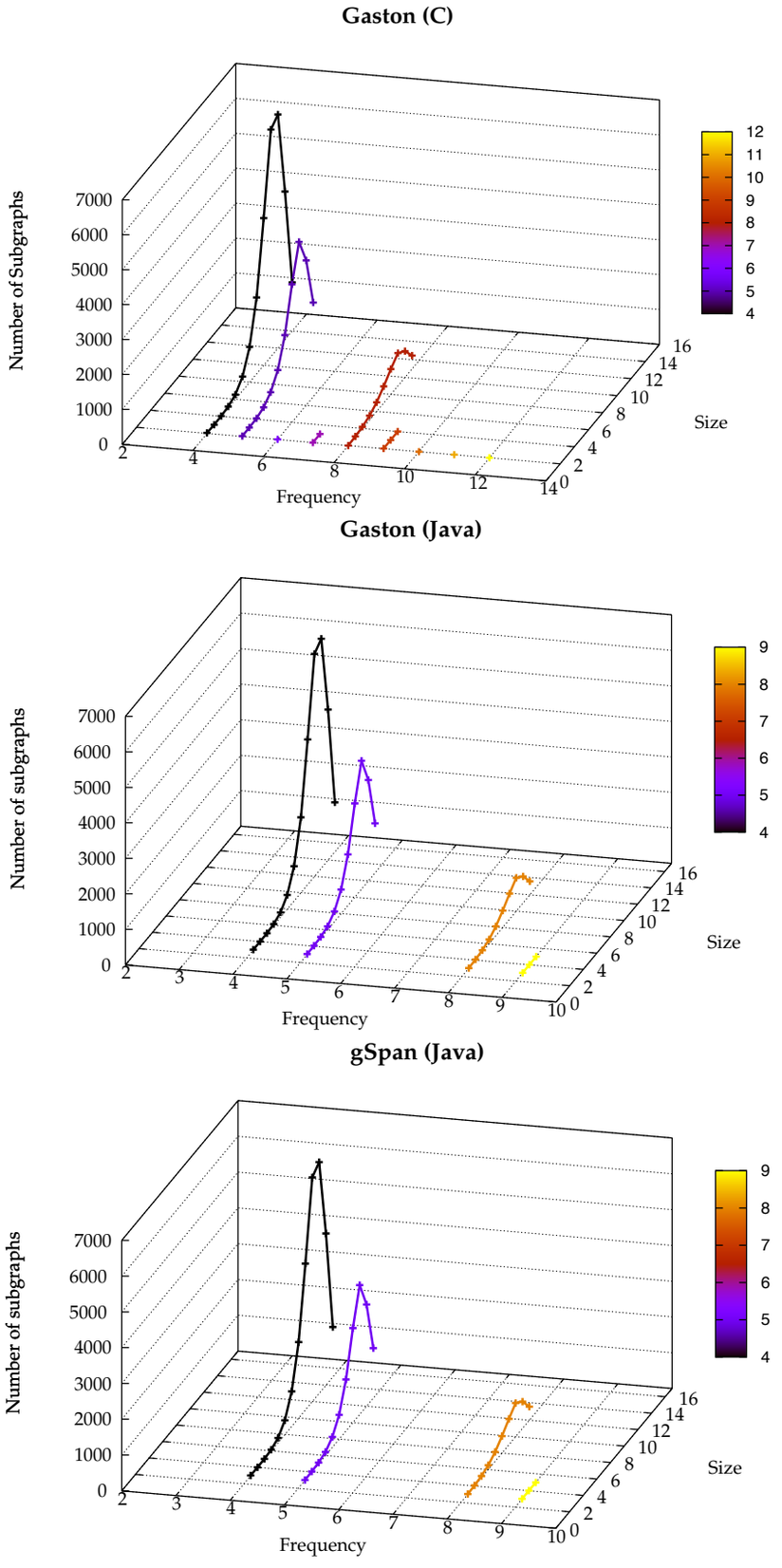


Figure 2.7: Evaluation results of different frequent subgraph mining algorithms.

4) Pattern-based systems focus on extracting patterns of specific components. It may be more useful to look at patterns in terms of the types of components that are assembled. Such an approach would incorporate semantic information with frequent patterns. Moreover, it would give a better idea of the overall operation that is being performed since quite often, a wholistic step consists of several operations. Thus far no workflow recommender has focused on these types of patterns. Some work has been done in this regard [68, 69]. This research shall investigate how such approaches can be adopted in this case.
5) Semantics-based systems cannot function until semantics about the components in the repository have been specified. Specifying the components semantically is a time-consuming task and requires concentrated efforts by users. In addition, semantics-based systems are static and do not learn as they are used. Therefore, unless the users put in the time and effort to specify the semantics of the components, the system will not be able to generate suggestions. Semantics-based systems are very good at determining which components are compatible with each other. However, since they do not incorporate frequent usage patterns, in situations where there are a number of similar components to choose from, they cannot prioritise the suggestions. Therefore, important expert knowledge encapsulated in the frequent usage patterns of other users is missing. In addition, two components may be semantically compatible; however the way they are implemented may make them unsuitable for use together. Semantics-based systems would be unable to determine this incompatibility unless it is explicitly specified.
6) Junaid et al [41] rank their suggestions according to design-time and runtime correctness. Design-time correctness includes the skill level of the user designing the workflow. However, a component is useful to a user if it allows that user to achieve the results he wants. Consequently, only those factors are useful to consider during suggestion generation that allow the system to determine the suitability of the component. The skill level of the user designing the workflow does not impact this suitability. Provided both users require the same results, an expert user would require the same components as a novice user. Therefore, incorporating this parameter into the suggestion ranking algorithm unnecessarily biases the suggestions. In addition, Junaid et al also incorporate the availability of past resources used by a component in the runtime correctness measure. This implies that the correctness of a suggestion is dependent upon the resources currently available in the grid. This assumes that the user would execute the workflow immediately after designing it. However, this might not be the case. The user might construct the workflow at one point, and then execute it at another. The available resources in the grid might change during this time. Therefore, the suggestions generated by the system would be misleading for the user. The system determines the compatibility between components on the basis of a weighted comparison of their attributes and provides some default values for the weights. These defaults have been determined experimentally. However, it is possible that different values for the weights in different scenarios would produce better results. The user can override the system defaults to suit their needs but this is an error-prone and time-consuming task. In light of these factors, the optimality of the suggestions can be questioned.
7) CAT is the only system that provides parameter suggestions; however the values are statically specified in the knowledge base. In most cases, parameter values are dependent on the data being provided to a component. Therefore, it might be more useful to have the system learn

parameter values based on usage patterns and suggest those to users.

8) Usability has been identified as an important aspect of any software, especially that which is intended to assist users. Some workflow composition systems such as Galaxy [22] have attempted to address specific usability challenges, but by and large it has been neglected. Since the focus of this research is to develop a prototype, user-oriented aspects such as usability are out of scope.

This research attempts to investigate the extent to which the aforementioned factors affect the suggestions in recommender systems. The goal shall be to address the aforementioned drawbacks.

2.11 Summary and Conclusions

This chapter describes the workflow composition process with the intention of identifying steps where intelligence can be applied to assist users when designing workflows. These include constructing correct workflows, finding appropriate components for their requirements, selecting parameter values, determining incompatibilities between components and identifying usage patterns in workflows for future suggestions. Most workflow management systems only address some of these challenges. They attempt to ensure workflow correctness by ensuring that input and output datatypes of connected components match. They generally lack sophisticated compatibility checks. In addition, most systems only provide simple text-based searching of components in the repository. However, some systems exist that add intelligent suggestion features to existing workflow composition systems. Such systems are termed as recommender systems in literature. This research classifies recommender systems into three categories according to the approach they adopt to generate suggestions for users and discusses the pros and cons of each category. The three categories are ratings-based systems, pattern-based systems and semantics-based systems. Rating-based systems employ explicit ratings for items provided by users and attempt to find other items that the user may be interested in. Patterns-based systems attempt to identify patterns in the way users use a system. They then suggest items to users based on these patters. Semantics-based systems attempt to suggest semantically-compatible items. It was concluded that pattern-based systems work for popular or frequently used items; but they omit rare items that may actually be more useful for the user. Semantics-based systems, on the other hand, are good at finding all items that the user may need, but cannot determine which of them are more appropriate in a particular scenario. Moreover, they require significant effort to set up initially.

One approach devised to reduce the effort required to enable the system to generate suggestions is to generate the semantics of the workflow components automatically [70]. This research attempts to address the same problem by combining features of both semantics-based and pattern-based systems. Instead of requiring the user to specify semantics by hand, they can only specify some basic descriptions. The system can then infer more semantics from the usage patterns of users and update the specified semantics. While doing so it is important to consider the expertise of the users from whom the patterns originate. Patterns originating from expert users should be given more weight than novice or intermediate users since it is likely the expert users are more aware of the different choices of workflow components and their tradeoffs than non-expert users. Moreover, context-aware parameter value suggestions are another important area where intelligent assistants can help. Since OWL 2 provides a rich set of features such as punning and qualified cardinality restrictions, it is the language chosen to represent the semantics of the workflow components in this research. For pattern

extraction, several algorithms were tested. Results-wise both Gaston and gSpan (Java and C implementations) perform similarly. The C implementation of Gaston was the most efficient, however the Java implementation of gSpan supports Closed graph mining [71]. It was also more efficient as compared to the Java implementation of Gaston. Therefore, gSpan has been chosen for the purposes of this research.

Based on the literature survey, Research Questions 1 and 3 have been answered. Limitations in existing workflow composition systems and related systems have been identified (Research Question 1) and suggestions to improve them have been formulated. Various subgraph mining algorithms were evaluated and gSpan was chosen for this research (Research Question 3). In Chapter 3 a novel approach combining both semantics and usage patterns is proposed. This will pave the way for fully answering Research Questions 2 and 4.

Bibliography

[1] S. P. Callahan *et al.*, "VisTrails: visualization meets data management," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 745–747. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142574

[2] T. Fahringer *et al.*, "ASKALON: a tool set for cluster and grid computing," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 143–169, 2005.

[3] I. Taylor *et al.*, "The Triana workflow environment: Architecture and applications," in *Workflows for e-Science*, I. J. Taylor *et al.*, Eds. Springer London, 2007, pp. 320–339, 10.1007/978-1-84628-757-2_20. [Online]. Available: http://dx.doi.org/10.1007/978-1-84628-757-2_20

[4] E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, 2014, in press.

[5] J. Ashburner *et al.*, "Computer-assisted imaging to assess brain structure in healthy and diseased brains," *Lancet Neurol*, vol. 2, pp. 79–88, Feb 2003.

[6] "AutoROIExtraction," http://bit.ly/1bJiZ9G.

[7] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Rec.*, vol. 34, no. 3, pp. 44–49, 2005. [Online]. Available: http://doi.acm.org/10.1145/1084805.1084814

[8] T. Tannenbaum *et al.*, "Beowulf cluster computing with windows," T. Sterling, Ed. Cambridge, MA, USA: MIT Press, 2002, ch. Condor: a distributed job scheduler, pp. 307–350. [Online]. Available: http://portal.acm.org/citation.cfm?id=571095.571112

[9] F. Berman *et al.*, "The GrADS project: Software support for high-level grid application development," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 327–344, November 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=1080645.1080669

[10] D. E. Rex, J. Q. Ma, and A. W. Toga, "The LONI pipeline processing environment," *NeuroImage*, vol. 19, no. 3, pp. 1033 – 1048, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105381190300185X

[11] E. Deelman *et al.*, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[12] Y. Gil *et al.*, "WINGS: Intelligent workflow-based design of computational experiments," *IEEE INTELLIGENT SYSTEMS*, vol. 26, no. 1, pp. 62–72, 2010.

[13] E. Deelman *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, Jul. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1239649.1239653

[14] J. Montagnat, T. Glatard, and D. Lingrand, "Data composition patterns in service-based workflows," in *Workshop on Workflows in Support of Large-Scale Science (WORKS'06)*, Paris, France, 2006. [Online]. Available: http://hal.inria.fr/inria-00615613

[15] T. Oinn *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/20/17/3045.abstract

[16] I. Altintas *et al.*, "Introduction to scientific workflow management and the Kepler system," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188669

[17] A. Goderis *et al.*, "Benchmarking workflow discovery: a case study from bioinformatics," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 16, pp. 2052–2069, 2009. [Online]. Available: http://dx.doi.org/10.1002/cpe.1447

[18] B. John, "A quantitative usability assessment method for inclusion in software engineering courses," in *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, May 2011, pp. 537–539.

[19] N. Bevan, "Extending quality in use to provide a framework for usability measurement," in *Human Centered Design*, ser. Lecture Notes in Computer Science, M. Kurosu, Ed. Springer Berlin / Heidelberg, 2009, vol. 5619, pp. 13–22, 10.1007/978-3-642-02806-9_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02806-9_2

[20] P. Lew, L. Olsina, and L. Zhang, "Quality, quality in use, actual usability and user experience as key drivers for web application evaluation," in *Web Engineering*, ser. Lecture Notes in Computer Science, B. Benatallah *et al.*, Eds. Springer Berlin Heidelberg, 2010, vol. 6189, pp. 218–232. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13911-6_15

[21] W. Albert and T. Tullis, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[22] J. Goecks *et al.*, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010. [Online]. Available: http://genomebiology.com/2010/11/8/R86

[23] P. M. Gordon and C. W. Sensen, "A pilot study into the usability of a scientific workflow construction tool," 2007.

[24] W. Tan *et al.*, "A comparison of using taverna and bpel in building scientific workflows: the case of cagrid," *Concurrency and computation : practice & experience*, vol. 22, no. 9, pp. 1098–1117, 06 2010. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2901112/

[25] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 734–749, 2005.

[26] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, pp. 76–80, 2003.

[27] B. N. Miller *et al.*, "MovieLens unplugged: experiences with an occasionally connected recommender system," in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, ser. IUI '03. New York, NY, USA: ACM, 2003, pp. 263–266. [Online]. Available: http://doi.acm.org/10.1145/604045.604094

[28] D. Billsus *et al.*, "Adaptive interfaces for ubiquitous web access," *Commun. ACM*, vol. 45, pp. 34–38, May 2002. [Online]. Available: w

[29] M. Pazzani and D. Billsus, "Learning and revising user profiles: The identification of interesting web sites," *Mach. Learn.*, vol. 27, pp. 313–331, June 1997. [Online]. Available: http://portal.acm.org/citation.cfm?id=261092.261098

[30] P. Resnick *et al.*, "GroupLens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ser. CSCW '94. New York, NY, USA: ACM, 1994, pp. 175–186. [Online]. Available: http://doi.acm.org/10.1145/192844.192905

[31] M. Claypool *et al.*, "Combining Content-Based and Collaborative Filters in an Online Newspaper," in *ACM SIGIR Workshop on Recommender Systems*, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.3659

[32] I. Soboroff and C. Nicholas, "Combining Content and Collaboration in Text Filtering," in *Proc. Int'l Joint Conf. Artificial Intelligence Workshop: Machine Learning for Information Filtering*, 1999.

[33] J. Crowell *et al.*, "A frequency-based technique to improve the spelling suggestion rank in medical queries." *Journal of the American Medical Informatics Association : JAMIA*, vol. 11, no. 3, 2004. [Online]. Available: http://ukpmc.ac.uk/abstract/MED/14764616

[34] "GSpell," http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/gSpell/current/GSpell.html [Last accessed: 22nd July, 2012].

[35] "Aspell," http://aspell.net/ [Last accessed: 22nd July, 2012].

[36] Z. Xu *et al.*, "Towards the semantic web: Collaborative tag suggestions," in *Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006*, 2006.

[37] D. Koop *et al.*, "VisComplete: Automating suggestions for visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1691–1698, Nov 2008.

[38] F. T. Oliveira *et al.*, "Provenance and annotation of data and processes," ser. Lecture Notes in Computer Science, J. Freire, D. Koop, and L. Moreau, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Using Provenance to Improve Workflow Design, pp. 136–143.

[39] E. Meij, P. Mika, and H. Zaragoza, "An evaluation of entity and frequency based query completion methods," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '09. New York, NY, USA: ACM, 2009, pp. 678–679. [Online]. Available: http://doi.acm.org/10.1145/1571941.1572074

[40] S.-S. Weng and M.-J. Liu, "Feature-based recommendations for one-to-one marketing," *Expert Systems with Applications*, vol. 26, no. 4, pp. 493 – 508, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740300188X

[41] M. Junaid *et al.*, "Workflow composition through design suggestions using design-time provenance information," *2009 5th IEEE International Conference on E-Science Workshops*, pp. 110–117, 2009.

[42] J. Kim, A. Gil, and M. Spraragen, "A knowledge-based approach to interactive workflow composition," in *In Proceedings of the 2004 Workshop on Planning and Scheduling for Web and Grid Services, at the 14th International Conference on Automatic Planning and Scheduling (ICAPS 04)*, 2004.

[43] P. Maechling *et al.*, "Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment," *SIGMOD Record*, vol. 34, no. 3, Sep 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1084805.1084811

[44] R. Bergmann and Y. Gil, "Similarity assessment and efficient retrieval of semantic workflows," *Information Systems*, vol. 40, pp. 115–127, 2014.

[45] E. Chinthaka *et al.*, "CBR based workflow composition assistant," in *Services-I, 2009 World Conference on.* IEEE, 2009, pp. 352–355.

[46] S. Han, G. M. Lee, and N. Crespi, "Semantic context-aware service composition for building automation system," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 1, pp. 752–761, Feb 2014.

[47] N. Mehandjiev *et al.*, "Assisted service composition for end users," in *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, Dec 2010, pp. 131–138.

[48] J. A. G. Coria, J. A. Castellanos-Garzón, and J. M. Corchado, "Intelligent business processes composition based on multi-agent systems," *Expert Systems with Applications*, vol. 41, no. 4, Part 1, pp. 1189 – 1205, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417413006143

[49] S. Roy Chowdhury *et al.*, "Complementary assistance mechanisms for end user mashup composition," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International

World Wide Web Conferences Steering Committee, 2013, pp. 269–272. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2487919

[50] Y. Mehmood *et al.*, "A middleware agnostic infrastructure for neuro-imaging analysis," in *22nd IEEE International Symposium on Computer-Based Medical Systems*, aug. 2009, pp. 1–4.

[51] S. Smanchat *et al.*, "Scheduling multiple parameter sweep workflow instances on the grid," in *Proceedings of the 2009 Fifth IEEE International Conference on e-Science*, ser. E-SCIENCE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 300–306. [Online]. Available: http://dx.doi.org/10.1109/e-Science.2009.49

[52] T. R. Gruber, "A translation approach to portable ontology specifications," *KNOWLEDGE ACQUISITION*, vol. 5, pp. 199–220, 1993.

[53] F. Baader and W. Nutt, "The description logic handbook," F. Baader *et al.*, Eds. New York, NY, USA: Cambridge University Press, 2003, ch. Basic description logics, pp. 43–95. [Online]. Available: http://portal.acm.org/citation.cfm?id=885746.885749

[54] B. Gibaud *et al.*, "NeuroLOG: sharing neuroimaging data using an ontology-based federated approach," in *AMIA Annual Symposium Proceedings*, vol. 2011. American Medical Informatics Association, 2011, p. 472.

[55] L. Temal *et al.*, "Towards an ontology for sharing medical images and regions of interest in neuroimaging," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 766–778, 2008.

[56] B. Batrancourt *et al.*, "A core ontology of instruments used for neurological, behavioral and cognitive assessments," in *Proceedings of the Sixth International Conference on formal Ontology in Information Systems (FOIS 2010)*, 2010, pp. 185–198.

[57] G. Kassel, "Integration of the DOLCE top-level ontology into the OntoSpec methodology," *arXiv preprint cs/0510050*, 2005.

[58] V. Haarslev *et al.*, "The RacerPro knowledge representation and reasoning system," *Semantic Web*, vol. 3, no. 3, pp. 267–277, 2012.

[59] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A highly-efficient OWL reasoner," in *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008, pp. 26–27.

[60] E. Sirin *et al.*, "Pellet: A practical OWL-DL reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[61] D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," *Automated Reasoning*, pp. 292–297, 2006.

[62] M. Krötzsch, F. Simancik, and I. Horrocks, "A description logic primer," *arXiv.org*, vol. cs.AI, Jan. 2012.

[63] E. Prud'Hommeaux, A. Seaborne *et al.*, "SPARQL query language for RDF," *W3C recommendation*, vol. 15, 2008.

[64] H. Wang, S. Zhai, and L. Fan, "Query for semantic web services using SPARQL-DL," in *Knowledge Acquisition and Modeling, 2009. KAM'09. Second International Symposium on*, vol. 1. IEEE, 2009, pp. 367–370.

[65] V. Krishna, N. N. R. R. Suri, and G. Athithan, "A comparative survey of algorithms for frequent subgraph discovery," *Current*, vol. 100, no. 2, p. 190, 2011. [Online]. Available: http://www.ias.ac.in/currsci/25jan2011/190.pdf

[66] S. Nijssen and J. Kok, "A quickstart in frequent structure mining can make a difference," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 647–652.

[67] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *IEEE International Conference on Data Mining, 2002. ICDM 2003. Proceedings. 2002.* IEEE, 2002, pp. 721–724.

[68] A. Inokuchi, "Mining generalized substructures from a set of labeled graphs," in *Fourth IEEE International Conference on Data Mining, 2004. ICDM'04.* IEEE, 2004, pp. 415–418.

[69] B. Berendt, "Using and learning semantics in frequent subgraph mining," *Advances in Web Mining and Web Usage Analysis*, pp. 18–38, 2006.

[70] Y. Gil *et al.*, "A semantic framework for automatic generation of computational workflows using distributed data and component catalogues," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 23, no. 4, pp. 389–467, 2011. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/0952813X.2010.490962

[71] X. Yan and J. Han, "CloseGraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 286–295.

CHAPTER 3

HyDRA - A Hybrid Architectural Framework for Patterns and Semantics-based Recommender Systems

3.1 Introduction

In Chapter 2, several limitations of existing workflow recommender systems were identified. It was concluded that they either employ pattern-based or semantics-based suggestion generation strategies. Pattern-based strategies work well for frequently-used components, but not for rarely-used ones. Semantics-based systems, on the other hand, lack the dynamism and empiricism of pattern-based systems. This is because they use statically defined semantic descriptions of the workflow components. Additionally, existing workflow management systems either do not provide parameter value suggestions at all, or they use statically-defined default values [1].

This chapter presents HyDRA, a novel hybrid architecture for suggestion generation. HyDRA combines both patterns and semantics to leverage the benefits of both and overcome their individual drawbacks. To achieve this, it presents a semi-automated methodology for semantic specification and enrichment. This chapter describes the overall process and architecture. The specific design and algorithms developed to realise this architecture are presented in Chapter 4.

In existing systems, users have to manually specify workflow component semantics. There is no mechanism to automatically infer new semantics. VisComplete [2] calculates frequent paths in workflow graphs. However, at suggestion generation time it uses heuristics to expand these paths into trees. This increases the computational overhead during suggestion generation. Precomputing trees may be more efficient than precomputing just paths. Pattern-based workflow recommenders consider only patterns of specific components. However, it may be more useful to consider functional patterns of components instead. Functional patterns carry more information and are more intuitive than specific patterns.

Current semantics-based workflow recommender systems require considerable user effort to be able to offer suggestions. This is because they require users to manually specify semantics for all the workflow components. One way to alleviate this is to allow the system to automatically infer semantics. In addition to presenting a model that addresses the aforementioned issues, this framework

will help in answering Research Question 4, reproduced here as well:

Question 4. *How can workflow component semantics and historical usage patterns be combined to improve the suggestions?*

The suggestion generation process is reviewed in Section 3.2. Once the reader has developed an idea of the overall process, the hybrid suggestion generation architecture that is the novel contribution of this thesis is described in Section 3.3. The various components and their functions are also described. Section 3.4 concludes and summarises the chapter.

3.2 The Suggestion Generation Model

Existing systems use either semantics or patterns to generate suggestions. Both of these sources provide useful information. Semantics are a useful tool to determine inter-component and input/output data type compatibility. It should be more useful to incorporate aspects of both in a hybrid framework. Semantics provide a more or less static description of the components being used. Adding reasoning support adds some dynamism to the system. However, the reasoning is still based on static descriptions. Workflow components, on the other hand, keep evolving. New algorithms emerge and existing ones get updated. This requires that users update the semantic repository in order for the suggestions to be relevant. Usage patterns, on the other hand, provide empirical knowledge not encapsulated in statically-defined semantics. They capture expert knowledge acquired by users through experience. This knowledge includes combinations of components that work well together, or components that are combined to perform a composite function or one not initially foreseen. This is a different aspect of knowledge than that provided by semantics, but an equally useful one. Such knowledge can conceivably be encoded into the semantic descriptions. However, the job of maintaining such a semantic repository would be both tedious and time-consuming. It would take up a considerable amount of a domain expert's time and effort that might be better spent conducting experiments and collecting and analysing results.

Because of these reasons, it might be more useful to combine both sources of knowledge. Combining them may allow the system to leverage the benefits of both aspects. Once some basic semantics for the components have been defined, they can be updated based on the patterns that are observed in practice. Additionally, patterns combined with semantics can allow the system to discover functional groups of components, which can help enhance the knowledge base. Moreover, once new components are added to the repository, it might be possible to infer their semantics from their context of use. Based on these observations, it is clear that a hybrid framework is required that incorporates both aspects of knowledge. This thesis presents such a hybrid model, described in detail below. Since the idea is to use patterns to inform and update the semantic descriptions, the semantic repository is the central entity in this model. As such, the suggestions are generated from this repository instead of directly from the patterns.

HyDRA's suggestion generation model is shown in Figure 3.1. Users use a *workflow management system* to compose their workflows. These workflows, once composed are stored in the *workflow repository*. The central entity in this process is the *domain ontology*, which contains the extracted patterns as well as semantics about the workflow components. These semantics include functional workflow

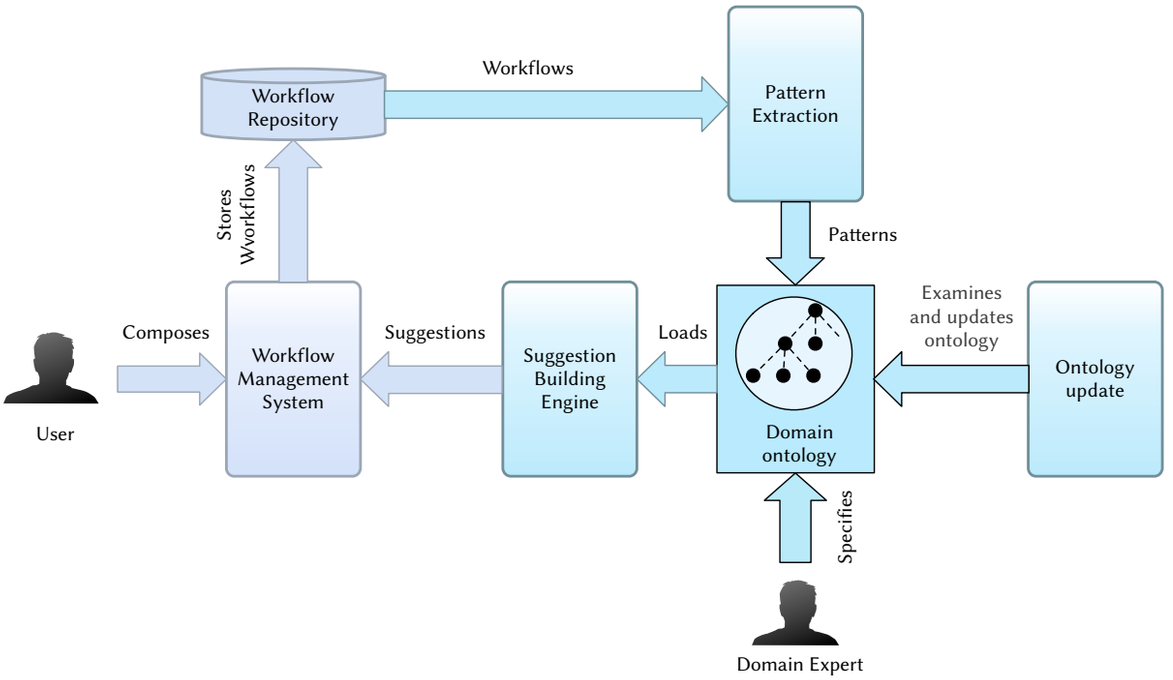


Figure 3.1: The suggestion generation process.

units that consist of several workflow components grouped into a logical unit. As mentioned previously, this framework employs a semi-automated methodology for semantic specification. The initial classification of components along with a description of their inputs and outputs is specified by domain experts. Additional semantic inference such as logical grouping of components is performed by the *ontology update* module based on the usage patterns observed. This module also detects new components and tries to infer their semantic properties from the context of use. The *pattern extraction* module is responsible for extracting the patterns from the workflows stored in the workflow repository. It performs several functions as well. These include (a) filtering unreliable workflows from the pattern extraction process, (b) preprocessing workflows for pattern extraction, and (c) generalising the components in order to extract functional patterns When suggestions are to be presented to a user, the the *suggestion building engine* extracts the suggestions from the ontology, formats, reorders and presents them to the user.

3.3 Hybrid Suggestion Generation Framework Architecture

The various steps described in the previous section perform distinct, but ultimately interconnected functions with concrete outputs. Moreover errors and failures may occur during their processing. Traceability of the results is also an important part of the scientific process. In order to facilitate error-tracking, maintainability and traceability, a modular architecture is required for HyDRA. In order to provide traceability, the architecture must provide several features such as verification and validation of the extracted patterns, verification of the domain ontology design and verification of the suggestions generated. As with traceability, maintainability can also be supported by providing different features. These include independent specification and administration of the domain ontology and the tweaking and configuration of the various modules. For error-tracking, each of the steps can be encoded into

distinct components with tangible inputs and outputs. Such a flexible, modular architecture is shown in Figure 3.2. It is described in detail in the following sections.

3.3.1 User Action Monitor

In order to be able to determine when suggestions should be presented to the user, monitoring user actions is required. There are three scenarios in which suggestion generation should be triggered: (a) when a new component is dropped on the canvas, prompting additional component suggestions, (b) when the user explicitly requests suggestions, and (c) when a link is specified between two existing components. This component is responsible for monitoring user actions while designing workflows. When a suggestion needs to be made, it contacts the Suggestion Building Engine (SBE) and sends the partially constructed workflow as part of the request. As response, it receives the list of suggestions from the broker, after which it formats and presents them to the user. The scenarios trigger a Component Suggestion Request (CSR). The request is sent to the Suggestion Request Broker (SRB) for further processing.

3.3.2 Suggestion Request Broker

This component acts as a single point-of-contact and liases between the various components within the SBE. It takes the partially constructed workflow from the User Action Monitor (UAM) and sends it first to the Semantic Analyser (SA) component and retrieves the analysed workflow. It then passes the workflow on to the Component Suggestion Request Handler (CSRH). Once it receives the suggestions from these components, it passes it on to the UAM.

3.3.3 Semantic Analyser

In order to generate suggestions, the partially constructed workflow must be semantically analysed to determine which components/patterns can be suggested. This involves propagating semantics across components. One such example is shown in Figure 3.3. The Cerebro algorithm performs segmentation of the cerebellum. It requires a skull-stripped brain image as shown in Figure 3.3a. The Bias Field Corrector tool, as mentioned in Chapter 2, performs error correction. As described previously, sometimes gradual errors occur in the recorded intensity values of the various voxels. This gradual variation is called the bias field of the image. Since it is simply an error removal tool, it requires an MRI scan as input. This component does not change the nature of the MRI however. Therefore, if a skull-stripped brain image is provided as input, it will output a skull-stripped brain image with error correction. Hence, Cerebro's requirement of a skull-stripped brain image can be fulfilled by attaching a component that produces such an image to the input Bias Field Corrector component. The system can determine this by propagating this requirement across components as shown in Figure 3.3b. The inferred semantics are added as annotations to the workflow and the annotated workflow is returned to the SRB.

3.3.4 Component Suggestion Request Handler

When suggestions to complete partial workflows are to be made, this component is invoked. It takes the semantically annotated partial workflow and uses a mix of OWLAPI calls and SPARQL-

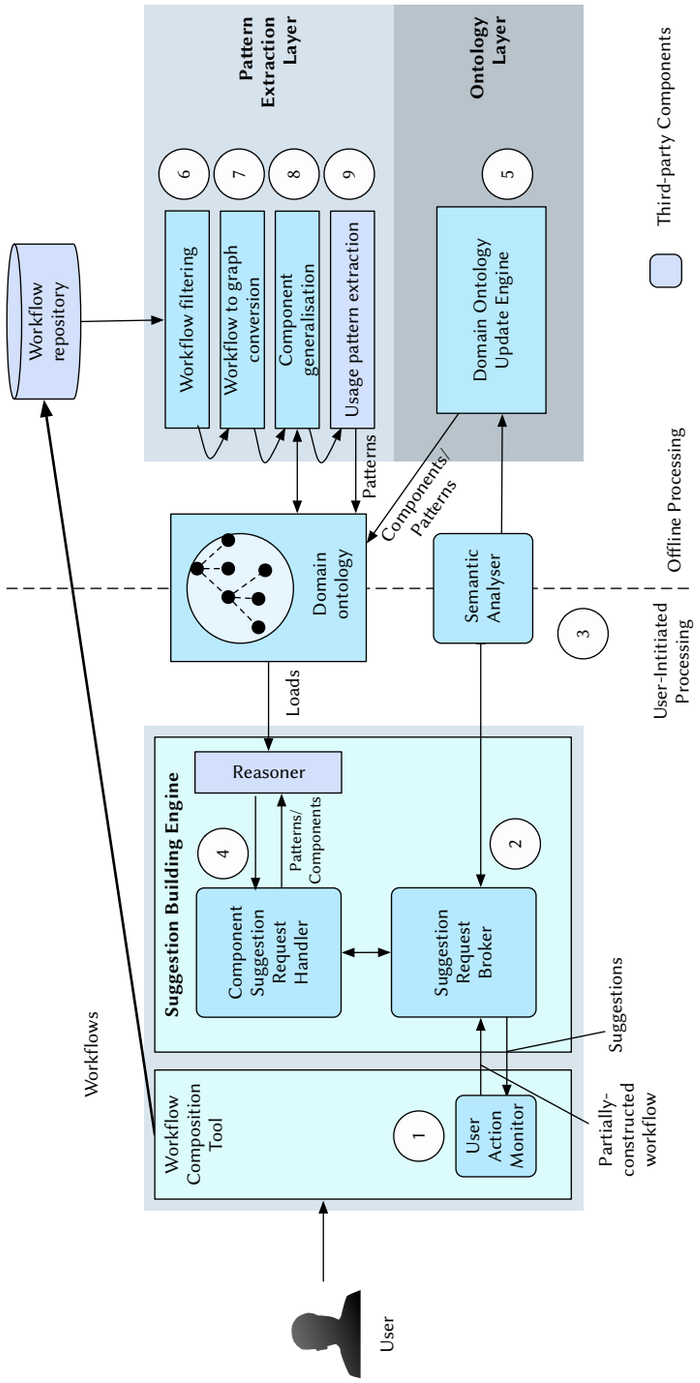
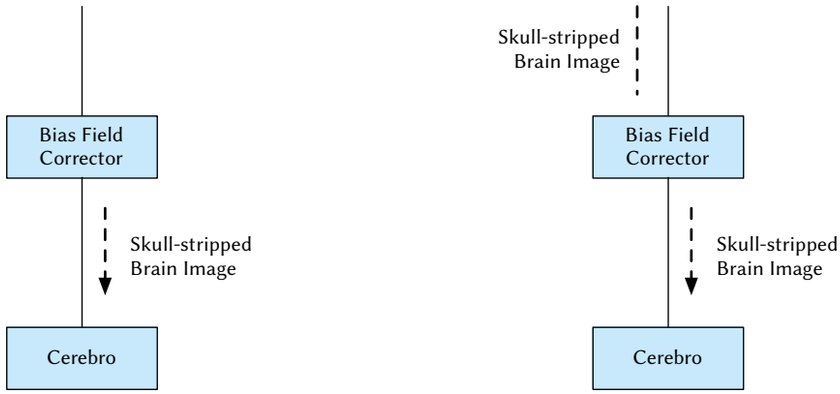


Figure 3.2: Hybrid suggestion generation framework architecture.



(a) Component requires skull-stripped brain image.

(b) Requirements are propagated across components.

Figure 3.3: A partial neuroimaging workflow.

DL queries to retrieve viable suggestions from the ontology via the Reasoner. The suggestions are returned to the SRB after reordering and pruning. The suggestions are generated from the discovered generalised patterns stored in the ontology. Existing systems that use patterns generate suggestions directly from the specific patterns. However, such systems do not work if a user is attempting to construct a workflow with new components. Since the components are new, little to no patterns exist that contain them. Looking at the patterns from a functional perspective allows the system to predict what the user is attempting to do and suggest patterns accordingly. In such a scenario, even if the specific components are new but perform a well-known existing function, the system would be able to assist the user. Alternatively, if the components perform a novel function, the system might still be able to suggest relevant components. However, the system might not be able to accurately predict what a user is attempting to do. In such a case, the relevant components might be suggested based on semantics, but the ordering might not be optimal. The following algorithm is used for these purposes:

1. The partial workflow is generalised to represent a partial pattern as represented in the ontology. The generalisation rules are discussed in Section 3.3.8.
2. All generalised patterns that contain the requested pattern as a subgraph are retrieved from the ontology.
3. The generalised components in the suggested patterns are specialised. During specialisation, many combinations of specific components are possible. Only combinations that are semantically valid and contain the requested partial workflow as a subgraph are shortlisted.

The shortlisted suggestions are sent back to the SRB.

3.3.5 Domain Ontology Update Engine

One of the major limitations in existing workflow recommenders concluded in Chapter 2 was that existing pattern-based systems use static semantic descriptions. The semantic repositories have to be specified, maintained and updated manually by users. This makes the job of the user very tedious and time-consuming. Additionally, the semantic descriptions only specify which components are compatible. There is no information about degrees of appropriateness; that is which components perform a

logical function together. Moreover there is no information about which ones are more appropriate for a particular task. Various systems have adopted different approaches to automatic/semi-automatic ontology generation. Some approaches exploit the inherent structure in existing data [3, 4, 5]. Others use text mining to extract structured knowledge from relevant literature [6, 7]. However, these approaches are not suitable for workflows. Often, the semantic descriptions about components are present in the form of free form text descriptions. There is no structure to the data. Additionally, there is no information about logical component groupings. Text mining approaches are also not suitable because quite often, the literature describing the components is sparse. Such approaches are also not suitable for extracting dependencies between the components. In workflows, usage patterns are a dynamic source of information. They provide logical component grouping as well as insight into best practices. Therefore, this thesis utilises usage patterns combined with user intervention to update and maintain the domain ontology. In this regard, the methodology is semi-automated. It is discussed in the following paragraphs.

This component is responsible for: (a) identifying new composite functional units of components, and (b) inferring semantics of new components. Figure 3.4 shows a partial workflow first introduced in Section 2.1. In this workflow, the component FLIRT is an alignment algorithm that estimates a trans-

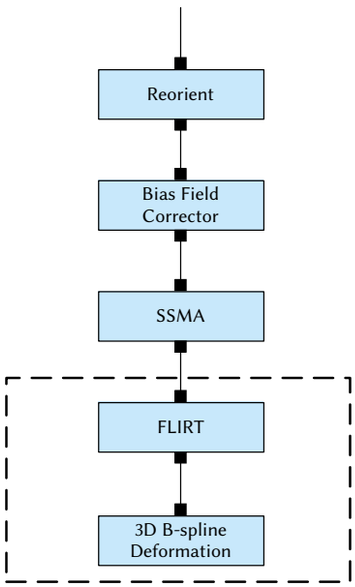


Figure 3.4: A functional unit.

formation between two images. The transformation matrix describes how the coordinate space of the source image can be transformed to match the coordinate space of the target image. The transformation matrix thus produced is used by 3D B-spline Deformation to actually perform the transformation, or "reslicing" as it it is also called. Consequently, these two components together represent a functional unit that represents a complete registration operation between two components. This unit is highlighted by the red border in Figure 3.4. Due to the fact that they represent a logical unit, it is likely these two components will appear recurrently. Therefore, they can be picked up during the usage pattern extraction process. Once this pattern makes it into the pattern repository, it is identified by this component as a logical unit which can then be written back to the ontology as one.

The other kind of update performed by the component is related to adding new components. If a

component appears in the patterns that is not present in the ontology, then it might be possible to infer its semantic properties from the context it is used in. Consider the workflow shown in Figure 3.5. This

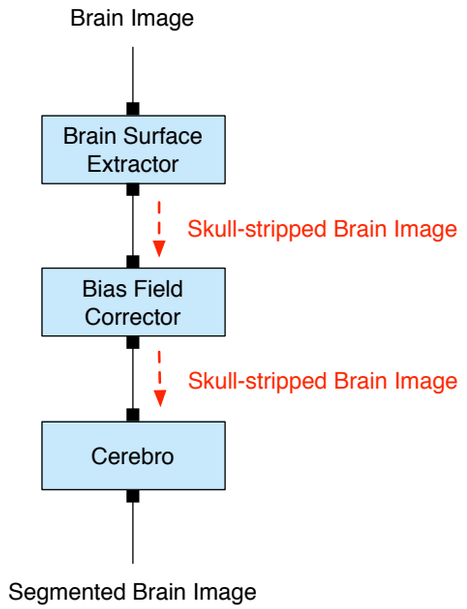


Figure 3.5: Inferring new component semantics.

workflow takes a brain image as an MRI, strips the skull and exposes the brain tissue (Brain Surface Extractor), performs error correction (Bias Field Corrector) and segments the cerebellum in the image (Cerebro). The output is saved in the target directory. Suppose the Brain Surface Extractor component has been newly added in the repository and the semantics have not yet been defined in the ontology. As discussed in Section 3.3.3, Cerebro requires a skull-stripped brain image. Suppose that it is known that the image provided as input to the workflow (represented in Figure 3.5 by the grey circle labelled "Brain Image") is not skull-stripped. Upon semantic analysis of the workflow (Section 3.3.3) it can be determined that the unknown component must produce such an image. Given that the unknown component takes a brain image, and produces a skull-stripped brain image, it can be reasonably inferred that it represents a skull-stripping algorithm. Using the aforementioned techniques, this component updates and adds to the ontology.

3.3.6 Workflow Filtering

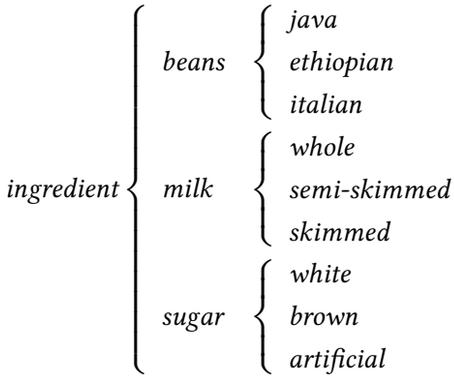
Some workflow filtering is performed in order to ensure that the learned patterns are useful and meaningful. Such workflows may arise due to various reasons. It may be because some users may just be playing around and exploring different options. The workflows may not perform any meaningful function. Novice users are more likely to create such workflows. The workflows may not even be correct due to incorrect parameter settings or improper connections. Moreover, they may be syntactically correct but may fail on the infrastructure due to various reasons such as inter-component incompatibilities. Including such workflows in the pattern extraction process will bias the suggestions towards incorrect/unreliable workflows. Therefore, they are filtered by this component.

3.3.7 Workflow-to-graph Conversion

In order to mine the workflows for patterns, they are converted to graphs for further processing. Reasons for this are discussed in Section 3.3.9. During this process, components and modules in the workflow become nodes in the graph. They are represented by different shapes to facilitate human readability. Links between components become edges in the graph. Finally, input and output data ports of modules are represented by labelling the edges in the graphs.

3.3.8 Component Generalisation

In order to identify functional units in the workflows, it is more appropriate to look at the various components in terms of the functions they perform rather than the specific algorithms they represent. For this reason, a functional taxonomy is used to generalise the workflow components to functional components. This taxonomy is encoded into the domain ontology. Algorithms that employ taxonomic context in this manner have been developed [8, 9]. One challenge in applying such a generalisation is the extent to which the components should be generalised. For example, suppose a taxonomy of ingredients is being used to analyse the coffee-making habits of customers with the taxonomy:



Now, a typical database of coffee recipes might contain patterns such as {*java*, *whole*, *white*, 4} and {*ethiopian*, *skimmed*, *brown*, 10}. The numbers 4 and 10 represent the frequencies with which these patterns occur. However, instead of extracting these patterns, it might be more meaningful and useful to extract the generic pattern {*beans*, *milk*, *sugar*, 14}. Doing so would allow the system to infer that an item containing these three general ingredients is popular amongst customers, which in this case is coffee. Notice that the frequency of the generalised pattern is greater than both the individual patterns. However, this may not always be the case. Suppose that the data only contained the pattern {*java*, *skimmed*, *white*, 4}. In this case, generalising it to {*beans*, *milk*, *sugar*, 4} yields a pattern with the same frequency. Generally, the frequency of occurrence of a generalised pattern will always be greater than or equal to the frequency of the specific patterns.

In the previous example, the ingredients could just as easily have been generalised to {*ingredient*, *ingredient*, *ingredient*}. However, generalising them in this manner is not useful since the pattern {*ingredient*, *ingredient*, *ingredient*, 14}, no matter how frequent, provides no meaningful information. Inokuchi [9] has proposed the pruning of such over-generalised patterns. According to his solution, for patterns P_1 and P_2, where P_1 is a generalisation of P_2, if $freq\{P_1\} = freq\{P_2\}$, then P_1 is an over-generalisation of P_2. However, this solution does not prune all over-generalised patterns. Suppose in

the previous example, if there were one more pattern in the data {*ethiopian*, *whole*, *artificial*, 4}, then generalising it to {*ingredient*, *ingredient*, *ingredient*, 4} would result in boosting the frequency of that pattern to 18. This pattern, according to Inokuchi's solution would be retained since it satisfies his criterion. However, it does not provide any useful information. Therefore, it would be better to prune it. In order to achieve this, this research employs a different generalisation strategy, discussed in the following paragraph.

It is clear from the preceding example, that the least general taxonomic terms that still differentiate between the various items in the pattern provide useful information. Therefore, this idea is adopted by this component to generalise the patterns. The following rules are observed during this process:

1. The algorithm starts with the most general term in the taxonomy that generalises all the various items.
2. The various items are incrementally specialised until no two items are represented by the same term.

Following these two steps, the algorithm arrives at the least general terms that generalise the various items while still retaining useful information. However, these rules are only suitable for sets of items with no specific links between them. Since workflows consist of components with explicit links between them, the rules have to be slightly modified in the following manner. Keep in mind that generalisation occurs after the workflows have been converted to graphs. Therefore, they are treated as graphs by the rules:

1. The algorithm starts with the most general node (component) in the taxonomy that generalises all the nodes in the graph (workflow).
2. All adjacent nodes that are represented by the same ancestor (general component) are iteratively specialised until they become distinct.

The criterion of adjacent nodes is introduced because a workflow may contain more than one of the same type of component. However, no two components of the same type will be directly connected via their inputs and outputs. In this manner, the graph representing a workflow is generalised to the least general components that still differentiate between the various components and thus retain useful information. The workflow graph is now ready for pattern mining.

3.3.9 Usage Pattern Extraction

Usage patterns are an important source of empirical knowledge. They capture expert knowledge acquired by scientific users with time and experience. In this framework, usage patterns are used to update and enrich the domain ontology. In order to do so, they must be extracted from the workflows for further processing. Pattern recognition is an appropriate technique that can be employed to do so.

Pattern recognition is a well-known problem in the field of artificial intelligence. It involves identifying different sorts of entities in a given dataset. These entities could be objects or patterns in the data. For example, Murty et al [10] have designed an algorithm to assist libraries in managing their collections. The algorithm organises the books into various groups based on their contents. These groups then facilitate the retrieval of books related to a particular topic as well as the classification of newly-added books. This example illustrates two significant problems in pattern recognition; clustering and classification. Clustering is the process of grouping a number of data points into distinct or fuzzy clusters. These clusters represent patterns in the data. Classification is the process of assigning

a particular data point to one of a number of precomputed clusters. In the quoted example, the process of separating the books into groups is clustering. The process of of assigning a newly-added book to one of existing groups is classification. Many algorithms that perform these tasks have been developed such as k-means, decision trees and bayesian networks [11, 12, 13] etc. Fundamentally, pattern recognition approaches rely on the similarity between the different points based on a number of features. Therefore, a significant challenge in pattern recognition is to identify features relevant to the problem that is being targeted. For example, in the library example the authors use the similarity between the tables of contents of the various books to cluster them. However, the patterns in workflows are of a slightly different nature. They consist of components that are not similar; they perform different, but complementary functions. Therefore, similarity is not a good metric to identify patterns in workflows. Instead, this thesis uses frequency of co-occurrence.

The inherent graph-like structure of workflows lends them to graph processing algorithms. The problem of identifying frequent patterns in graphical workflows can be represented as finding frequent subgraphs in a set of graphs. Frequent subgraph mining is a well-researched problem and many algorithms exist. Therefore, this thesis uses an existing subgraph mining algorithm for finding frequent patterns in workflows. The usage pattern extraction component is responsible for extracting the patterns from the generalised workflow graphs and storing them in the repository. As discussed in Section 2.11, several algorithms were tested for this purpose and gSpan [14] was chosen. The reasons for this have also already been discussed. The output of this component is a set of frequent functional workflow units that represent functional usage patterns. The following section summarises and concludes this chapter.

3.4 Summary and Conclusions

In this chapter HyDRA, a novel hybrid framework for suggestion generation was proposed. It employs both frequent usage patterns and workflow component semantics to generate suggestions. At the core of this framework is a domain ontology which contains both the semantics and learned patterns. The framework employs semi-automated methodologies to update and enrich the ontology based on the learned usage patterns. The framework consists of two distinct phases. One is the user-initiated processing phase and the other is the offline processing phase. In the user-initiated processing phase, the framework monitors the user while they compose a workflow. When the user is to be presented with suggestions, the framework uses the information stored in the ontology to generate them based on a semantic analysis of the partially-constructed workflow. In the offline processing phase, the workflow repository is analysed to extract frequent usage patterns. These patterns are then used to identify new functional units as well as new workflow component semantics.

The framework proposed in this chapter helps us in answering Research Question 4 as it combines both semantics and frequent usage patterns. In the next chapter, the suggestions generation process is discussed in detail along with the various algorithms that are used by the different components. The details of the domain ontology are also discussed.

Bibliography

[1] J. Kim, M. Spraragen, and Y. Gil, "An intelligent assistant for interactive workflow composition," *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, Jan 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=964442.964466

[2] D. Koop *et al.*, "VisComplete: Automating suggestions for visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1691–1698, Nov 2008.

[3] W. Ahmed *et al.*, "A light weight approach for ontology generation and change synchronization between ontologies and source relational databases," in *15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2011.* IEEE, 2011, pp. 208–214.

[4] L. Cagliero, T. Cerquitelli, and P. Garza, "Semi-automatic ontology construction by exploiting functional dependencies and association rules," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 7, no. 2, pp. 1–22, 2011.

[5] N. Yahia, S. Mokhtar, and A. Ahmed, "Automatic generation of owl ontology from xml data source," *arXiv preprint arXiv:1206.0570*, 2012.

[6] C. Blaschke and A. Valencia, "Automatic ontology construction from the literature," *Genome Informatics Series*, pp. 201–213, 2002.

[7] R. Krishnan, A. Hussain, and P. Sherimon, "Retrieval of semantic concepts based on analysis of texts for automatic construction of ontology," in *Neural Information Processing.* Springer, 2012, pp. 524–532.

[8] B. Berendt, "Using and learning semantics in frequent subgraph mining," *Advances in Web Mining and Web Usage Analysis*, pp. 18–38, 2006.

[9] A. Inokuchi, "Mining generalized substructures from a set of labeled graphs," in *Fourth IEEE International Conference on Data Mining, 2004. ICDM'04.* IEEE, 2004, pp. 415–418.

[10] M. N. Murty and A. K. Jain, "Knowledge-based clustering scheme for collection management and retrieval of library books," *Pattern recognition*, vol. 28, no. 7, pp. 949–963, 1995.

[11] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2, pp. 131–163, 1997.

[12] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 281-297. California, USA, 1967, p. 14.

[13] D. Magerman, "Statistical decision-tree models for parsing," in *Proceedings of the 33rd annual meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 1995, pp. 276–283.

[14] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *IEEE International Conference on Data Mining, 2002. ICDM 2003. Proceedings. 2002.* IEEE, 2002, pp. 721–724.

CHAPTER 4

Realising HyDRA

The previous chapter presents HyDRA, a novel hybrid architecture for suggestion generation. This architecture employs semantics and frequent usage patterns to generate suggestions. The previous chapter discussed the various components that comprise HyDRA's framework and presented their rationale. The information presented gave an overall picture of the functions performed by HyDRA and how they were interconnected. In this chapter a prototype implementation of HyDRA is discussed. Various examples introduced in the previous chapter are used here to highlight how the functions discussed can be implemented. It is divided into two main parts. The first part discusses the various algorithms that perform the functions described in the previous chapter. The second part discusses the contents of the domain ontology, also called the knowledge base, and explains how they can be used to generate suggestions. The suggestion generation process involves several steps:

1. The partial workflow being constructed has to be semantically analysed to generate contextual suggestions.
2. The partial workflow being constructed has to be generalised so that the system may attempt to infer what a user is attempting to do based on the frequent usage patterns previously extracted.
3. The results of the previous two steps have to be combined to generate a unified list of suggestions.

In essence, semantics are applied in the first two steps to generate suggestions. This chapter discusses prototype implementations of all these steps.

Some essential differentiating terminology is discussed in Section 4.1. The various symbols and notations that are required for understanding this chapter and the information presented are also discussed in this section. Section 4.2 discusses the implementation of the semantic analyser along with a discussion of its inputs and outputs. This is followed by a discussion of the generalisation process and how it is achieved in Section 4.3. The final list of suggestions is compiled by the algorithm presented in Section 4.4. This concludes the first part of this chapter. Section 4.5 relates to the second part about the domain ontology and how the knowledge encoded in it can be used to support suggestion generation. Section 4.6 presents an illustrative example that conceptually demonstrates how HyDRA interacts with a user followed by conclusions in Section 4.7.

4.1 Essential Terminology

In order to understand this chapter, an introduction of some essential notations and terminologies is required. These notations and terminologies shall be used throughout this chapter as well as subsequent chapters. There are two types of entities discussed in this research; *workflows* and *ontologies*. Both entities have their own standard terminologies and concepts, introduced in this section for convenience. A workflow is an ordered sequence of steps that each perform some kind of data processing on datasets. Each step is called a *workflow component*. It represents an executable computer program or algorithm that performs the required processing. An *ontology*, on the other hand, is a knowledge base that contains knowledge about some real world entities. It defines the properties, roles and relationships between those entities. The components of an ontology are *concepts*, *roles* and *individuals*.

To discuss the representation of knowledge Description-Logic (DL) is used [1]. The rationale behind it is discussed in Section 4.5. Ontologies, and indeed DLs are used to model relationships between different entities. In DL, as in ontologies, there are three types of entities; concepts, roles and individuals. Concepts denote sets of individuals, while roles denote relationships between entities. Figure 4.1 shows an example ontology that can be modelled using DL. The ontology contains two concepts; *Parent* and *Child*. The relationship between them may be expressed via the role *has-child*. *Parent(Tom)* specifies that the individual *Tom* belongs to the concept *Parent*. *Parent* is also called the *type* of the individual *Tom*. *has-child(Tom,Jack)* states that the individuals *Tom* and *Jack* are related to each other through the role *has-child*. Similarly, *has-child(John,Jill)* specifies that John has a child named Jill. In

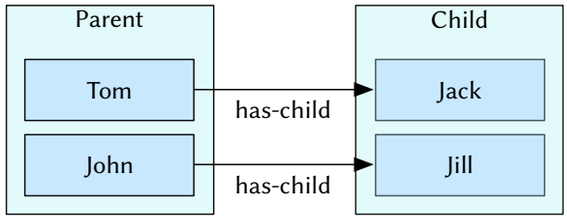


Figure 4.1: Description logic entities.

addition to these entities, DL also defines various constructs for defining complex relationships. For convenience, the various constructs and notations used in this research and their notations are defined here:

Subsumption	$(\sqsubseteq)$	A binary relationship that can exist between two concepts in an ontology, where one concept denotes a more specific set of individuals than the other. This construct can be used to describe a taxonomic hierarchy, such as the one this research requires. For example, $A \sqsubseteq B$ states that A is a specialisation of B. A is said to be *subsumed* by B.
Intersection	$(\sqcap)$	A binary relationship that specifies the intersection of two concepts. $A \sqcap B$ specifies all individuals that belong to both A and B.
Union	$(\sqcup)$	A binary relationship that specifies the union of two concepts. $A \sqcup B$ specifies all individuals that either belong to A or B.
Disjoint		A binary relationship that specifies that two concepts are mutually exclusive. $disjoint(A, B)$ specifies that no individual can belong to both A and B at the same time. In DL it can also be expressed as $A \cap B = \emptyset$.
Concatenation	$(\oplus)$	This research defines the $\oplus$ operation as a binary *concatenate* operation on two sets. For sets $s_1 = \{1, 2, 3\}$ and $s_2 = \{4, 5, 6\}$, $s_1 \oplus s_2$ results in $s_1 = \{1, 2, 3, 4, 5, 6\}$ and $s_2 = \{4, 5, 6\}$.
Difference	$(-)$	A binary difference operation on sets that subtracts the second set from the first one. This means that for the sets $s_1 = \{1, 2, 3\}$ and $s_2 = \{2, 3\}$, $s_1 - s_2$ gives $\{1\}$.

As discussed in Section 3.3.8, this research uses a taxonomy of workflow components. A taxonomy is essentially a mechanism of classifying and organising various entities in a particular domain. Therefore, intuitively an ontology can also act as a taxonomy. The various concepts of the ontology can serve as classifying branches of the taxonomy. Its hierarchical nature is ideal for representing the subsumption relationships in an ontology.

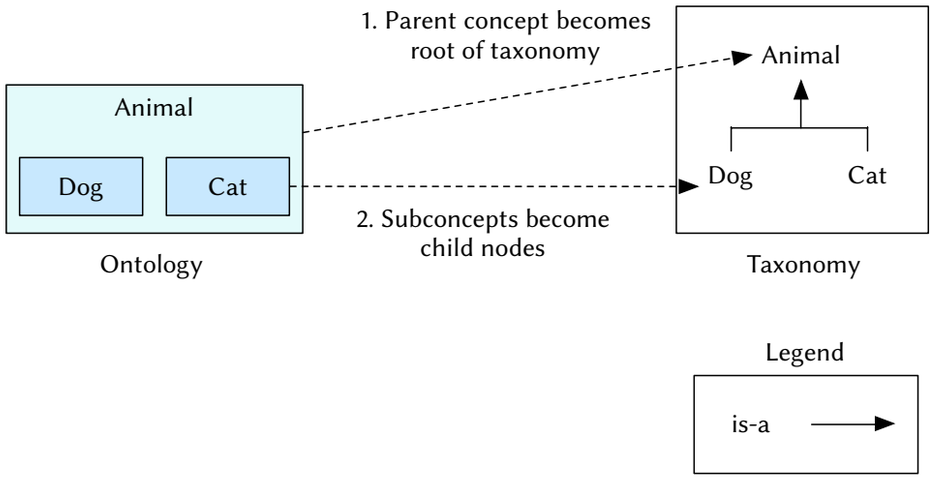


Figure 4.2: Ontology to taxonomy mapping.

Consider the example ontology shown in Figure 4.2. The concept *Animal* subsumes the subconcepts *Dog* and *Cat*. When converting this ontology to a taxonomy, the parent concept *Animal* becomes the root of the taxonomy. All the subconcepts become the children of the root in the taxonomy. More-

over, the relationships between the various concepts can be expressed using the following DL axioms:

$$\text{Dog} \sqsubseteq \text{Animal}$$
$$\text{Cat} \sqsubseteq \text{Animal}$$

These axioms state that *Dog* and *Cat* are specific types of *Animal*. Thus, the subsumption operator can be used to specify a taxonomy. This fact will be exploited in later sections.

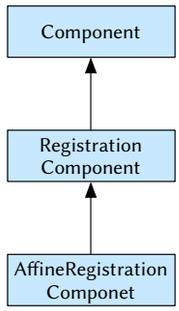


Figure 4.3: Example neuroimaging taxonomy.

Scientific workflows consist of various components that constitute the processing steps of the workflow. This research uses an ontology that classifies the workflow components by their function. For example, a workflow component called FLIRT, which is part of the FSL [2] suite of algorithms, performs a normalisation process called registration between MRIs. More specifically, there are various types of registration that can be performed; e.g. affine and non-affine registration. FLIRT is an affine registration algorithm. Therefore, one way to classify FLIRT is to define the concepts *Component*, *RegistrationComponent* and *AffineRegistrationComponent* in the ontology. The corresponding taxonomy of this ontology is shown in Figure 4.3. Given this taxonomy, the axiom *AffineRegistrationComponent(FLIRT)* states that FLIRT is an affine registration component that is also a registration component and a workflow component.

During the generalisation process (discussed in Section 4.3), the workflow components are often replaced by their taxonomic classes. In order to facilitate the discussion of this process and to avoid confusion, some terminology to distinguish between the actual workflow components and their taxonomic classes is introduced. Generally, the components are executable algorithms that process data in some way. In contrast, their taxonomic classes are non-executable components that represent generic functions. In the example discussed, FLIRT is an executable algorithm whereas the concepts *AffineRegistrationComponent*, *RegistrationComponent* and *Component* are its non-executable taxonomic classes. The terms introduced exploit this characteristic of the workflow components. They are borrowed from the work of Kim et al. [3]:

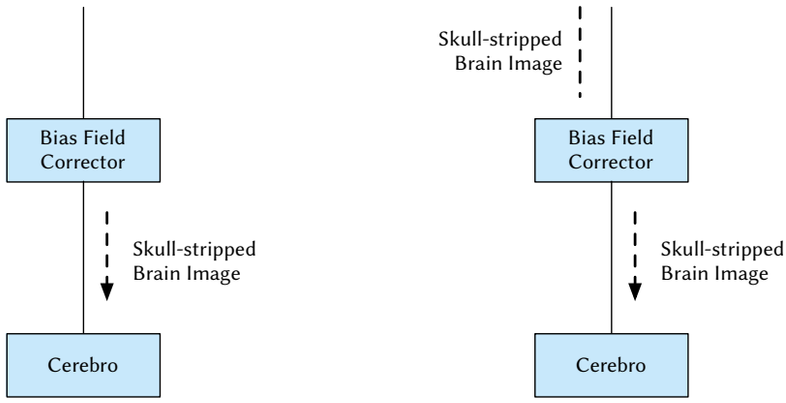
Abstract component: a component that is not executable. A workflow that contains at least one abstract component is called an *abstract workflow*.

Concrete component: a component that is executable. A workflow that contains no abstract components is called a *concrete workflow*.

According to these definitions, FLIRT is a concrete component while its taxonomic classes are abstract components. Having presented all of the preliminaries, the subsequent sections discuss the various functions that the framework performs and how they are implemented.

4.2 Semantically Analysing Workflows

In order to generate accurate suggestions for workflows, the semantic properties of workflow components need to be propagated across the workflow. To understand why this is necessary, an illustrative example has already been discussed in Section 3.3.3. The example is reintroduced here for convenience and for further expansion. Consider the two workflow components, *Cerebro* and *Bias Field Corrector*. *Cerebro* is a segmentation algorithm that identifies various parts of the brain and labels them accordingly. To achieve this, it requires a skull-stripped MRI, in which the brain tissue has been extracted from the skull and other materials. Often, an MRI contains noise that must be filtered before it can be reliably processed for further use. The *Bias Field Corrector* component is responsible for doing exactly that.



(a) Component requires skull-stripped brain image.

(b) Requirements are propagated across components.

Figure 4.4: A partial neuroimaging workflow.

Since the function of the *Bias Field Corrector* component is to reduce the noise in the MRI, it does not otherwise change the nature of image. If the MRI has already been skull-stripped, it will remain a skull-stripped image after it has been processed by the *Bias Field Corrector* component. Now consider the partial workflow shown in Figure 4.4. Figure 4.4a shows a partial workflow for which suggestions are to be generated. It can be seen that *Cerebro* requires a skull-stripped MRI. The *Bias Field Corrector* is not a skull-stripping component. So how can *Cerebro*'s requirement be fulfilled? It can be seen that if a skull-stripped image is provided as input to the *Bias Field Corrector*, it will produce a noise-reduced skull-stripped image. Because of this transitive behaviour of the *Bias Field Corrector*, generating accurate suggestions for a workflow involves propagating semantic properties as shown in Figure 4.4b.

In general, this process is depicted in Figure 4.5. Consider two workflow components c_i and c_m with inputs and outputs as shown with a link between them. In addition, there also exists a *transitive dependency* from c_{mn} to c_{mo}. That is to say, the workflow component c_m transfers the semantics properties from its input c_{mn} to its output c_{mo}. The previously discussed *Bias Field Corrector* component is an example of such a component. It can be seen from the figure that c_{mn} gets its semantic properties from c_{ij}, which are then transferred to the output c_{mo} because of the transitive dependency. If $\text{prop}(c_{ij})$ gives the semantic properties for c_{ij}, then the operation $\text{prop}(c_{mo}) \oplus \text{prop}(c_{ij})$

[File Creation Timestamp: 24/02/2016, 14:13]

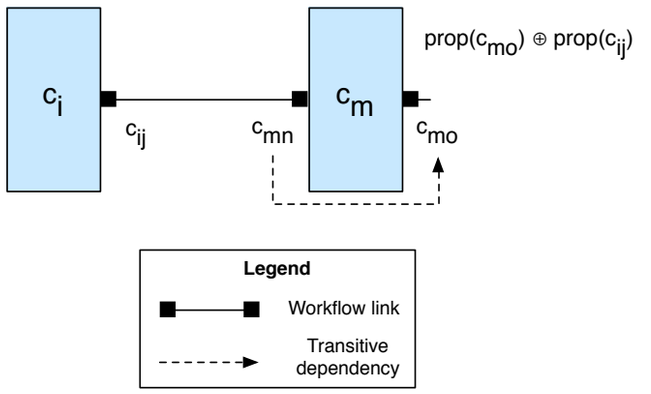


Figure 4.5: Semantic propagation when a transitive dependency exists

transfers those properties to c_{mo}. As an example, consider the workflow components *Brain Surface Extractor* and *Bias Field Corrector*, both part of the BrainSuite set of algorithms [4]. *Brain Surface Extractor* is a skull-stripping component that removes the skull and other extraneous tissue from an MRI and exposes the brain. It produces a skull-stripped image as a result. If c_i is replaced by *Brain Surface Extractor* and c_m by *Bias Field Corrector*, then semantic analysis will reveal that the output of *Bias Field Corrector* is also a skull-stripped image in this case. In order to properly propagate the semantic properties across the workflow, they must be both propagated forward and backward. Therefore semantic analysis involved two steps; forward propagation and backward propagation, explained in the following sections.

4.2.1 Forward Propagation

The first step in the semantic analysis process is forward propagation. In this step semantics are propagated from the inputs of workflow components to their outputs. Figure 4.5 is an example of forward propagation. Suppose that for a particular workflow, C is the set of all components in the workflow. L is the set of tuples of the form *<source,sink>* representing the links between the workflow components. σ is the set of tuples of the form *<source,target>* which represent the *transitive relationships* between the component parameters. This research defines a transitive relationship to exist between two parameters when the semantic properties of an input parameter are transferred to an output parameter. The *Bias Field Corrector* component is one such example.

In a transitive relationship the *target* is said to be *transitive dependent* on the *source*. However, workflow components can have multiple input and output parameters. This may give rise to some ambiguity as to which parameters are transitive dependent on each other. To resolve this ambiguity, this approach requires domain experts to specify transitive dependencies between the parameters of different components. These dependencies are encapsulated in the σ parameter. In Figure 4.5, c_{mo} is transitive dependent on c_{mn}. Therefore, for the component c_m, $\langle c_{mn}, c_{mo} \rangle \in \sigma$. For more details about transitive dependencies see Section 4.5.2.

During forward propagation, for each workflow component, the system must first determine if semantic propagation needs to be performed. This involves determining if, for each pair of connected output and input parameters, the input parameter has a transitive dependency. For example, in Figure 4.5, c_{ij} is connected to c_{mn}. Therefore, $\langle c_{ij}, c_{mn} \rangle \in L$. Recall that L is the set of links in the

[File Creation Timestamp: 24/02/2016, 14:13]

workflow. Moreover, c_{mo} is transitive dependent on c_{mn}, therefore $\langle c_{mn}, c_{mo} \rangle \in \sigma$. Given these two facts, the system can determine that semantic propagation needs to take place from c_{ij} to c_{mo}. The following conditional statement can be used to determine if semantic propagation needs to be performed:

$$\exists\, c_{ij}, c_{mn}, c_{mo} \mid c_{ij} \in \text{Output}(c_i) \;\wedge\; c_{mn} \in \text{Input}(c_m) \\ \wedge\; c_{mo} \in \text{Output}(c_m) \wedge\; \langle c_{ij}, c_{mn} \rangle \in L \;\wedge\; \langle c_{mn}, c_{mo} \rangle \in \sigma$$

This condition states that there exist some c_{ij}, c_{mn} and c_{mo} such that c_{ij} is an output parameter of c_i and c_{mn} and c_{mo} are input and output parameters of c_m respectively. Additionally, $\langle c_{ij}, c_{mn} \rangle \in L$ meaning c_{ij} is connected to c_{mn}. Finally, $\langle c_{mn}, c_{mo} \rangle \in \sigma$, indicating that there is a transitive dependency from c_{mn} to c_{mo}. If all of these conditions are found to be true, then the semantic properties of c_{ij} are to be transferred to c_{mo}.

Algorithm 1 Forward propagation algorithm.

Require: C : Set of components in the workflow.
L : Set of source-sink tuples representing the links in the workflow.
σ : Set of origin-target tuples representing the transitive dependencies between inputs and outputs of components.

1: **procedure** ForwardPropagation(C, L, σ)
2: **for all** $c_i \in C$ **do**
3: **for all** $c_{ij} \in \text{Output}(c_i)$ **do**
4: **if** $\exists\, c_{mn}, c_{mo} \mid c_{mn} \in \text{Input}(c_m), c_{mo} \in \text{Output}(c_m) \;\wedge\; \langle c_{ij}, c_{mn} \rangle \in L \;\wedge\; \langle c_{mn}, c_{mo} \rangle \in \sigma$ **then**
5: $\text{prop}(c_{mo}) \oplus \text{prop}(c_{ij}) \mid \nexists\, p_i, p_j \;:\; p_i \in \text{prop}(c_{ij}), p_j \in \text{prop}(c_{mo}) \wedge\; p_i \sqcap p_j = \emptyset$
6: **end if**
7: **end for**
8: **end for**
9: **end procedure**

The following statement transfers the semantic properties from c_{ij} to c_{mo}:

$$\text{prop}(c_{mo}) \oplus \text{prop}(c_{ij})$$

However, care must be taken at this point as this step may give rise to sets of conflicting properties. For example, consider the situation where $\text{prop}(c_{ij}) = \langle \text{UnsegementedDataset} \rangle$ and $\text{prop}(c_{mo}) = \langle \text{SegmentedDataset} \rangle$. This situation arises in the case of a workflow component that takes an unsegmented MRI and produces a segmented one such as *Cerebro*. In this case, transferring the semantic properties from c_{ij} to c_{mo} will result in $\text{prop}(c_{mo}) = \langle \text{UnsegmentedDataset}, \text{SegmentedDataset} \rangle$. This is absurd because no image can be both unsegmented and segmented at the same time. This situation can be avoided by declaring these two properties as *disjoint*. All that is then required is to add an additional check when propagating the semantics to ensure disjoint properties are excluded. The following statement achieves this purpose:

$$\text{prop}(c_{ij}) \sqcap \text{prop}(c_{mo}) = \emptyset$$

This states that $\text{prop}(c_{ij})$ and $\text{prop}(c_{mo})$ do not contain any properties that are mutually disjoint. The complete algorithm is shown in Algorithm 1.

The algorithm iterates over all of the components in the workflows (Line 2). Each component c_i has some input and output parameters denoted by c_{ij}. Input parameters are denoted by $\textit{Input}(c_i)$.

Similarly, output parameters are denoted by *Output*(c_i). For each output parameter of each component, the algorithm first checks if it is connected to the input parameter of another component (Line 4). If there exists a transitive dependency between that input parameter and an output parameter of the same component, the semantics are propagated (Line 5). The process continues until all semantics have been propagated across the workflow.

4.2.2 Backward Propagation

The second step in the semantic analysis process is backward propagation of semantic properties. This is equivalent to determining what the semantic requirements for a partial workflow are since these have to be satisfied by some additional workflow components. The backward propagation process mirrors the forward propagation process. It is depicted in Figure 4.6. In this case, c_{ij} is an input

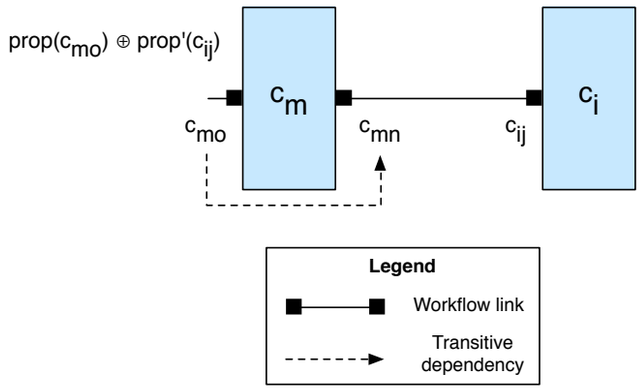


Figure 4.6: Backward propagation.

to c_i with specific semantic properties. These properties have to be satisfied by c_{mn} since the two are connected. Moreover, there is a transitive dependency between c_{mo} and c_{mn}. This means that any semantic properties available at c_{mo} will be propagated to c_{mn}. Therefore, the semantic requirements for c_{ij} can also be satisfied by c_{mo} if they are not satisfied by c_{mn}. This is the key difference between forward propagation and backward propagation. The semantic properties are back-propagated iff they have not been satisfied. Otherwise, there is no point. The example shown in Figure 4.4 depicts backward propagation in practice.

Formally, the following conditional statement can determine whether semantic propagation should take place:

$$\exists\, c_{ij}, c_{mn}, c_{mo} \mid c_{ij} \in \text{Input}(c_i) \;\wedge\; c_{mn} \in \text{Output}(c_m) \\ \wedge\; c_{mo} \in \text{Input}(c_m) \;\wedge\; \langle c_{mn}, c_{ij} \rangle \in L \;\wedge\; \langle c_{mo}, c_{mn} \rangle \in \sigma$$

This states that there should be some workflow components c_i with input c_{ij} and c_m with output c_{mn} and input c_{mo}. There should also be al link between c_{mn} and c_{ij} and a transitive dependency between c_{mo} and c_{mn}. If these conditions are satisfied, then semantic propagation needs to be performed.

Similarly to forward propagation, the operation $\text{prop}(c_{mo}) \oplus \text{prop}(c_{ij})$ will transfer the semantic properties from c_{ij} to c_{mo}. However, as mentioned previously, only semantic properties that have not been satisfied must be transferred. Those that have been satisfied should be excluded from this. Now

[File Creation Timestamp: 24/02/2016, 14:13]

the semantic properties of c_{ij} are satisfied by those of c_{mn}. Therefore, the system must check if there are any common properties between c_{mn} and c_{ij} as those will be the ones that have been satisfied. The final operation can be written as:

$$\text{prop}(c_{mo}) \oplus \text{prop}'(c_{ij}) \mid \text{prop}'(c_{ij}) \leftarrow \text{prop}(c_{mn}) - \text{prop}(c_{ij})$$

This states that transfer all properties from c_{ij} to c_{mo} that are not also properties of c_{mn}. The complete algorithm is given in Algorithm 2. Similar to forward propagation, the algorithm iterates over all the

Algorithm 2 Backward propagation algorithm.

Require: C : Set of components in the workflow.
L : Set of source-sink tuples representing the links in the workflow.
σ : Set of origin-target tuples representing the transitive dependencies between inputs and outputs of components.

1: **procedure** BACKPROPAGATION(C, L, σ)
2: **for all** $c_i \in C$ **do**
3: **for all** $c_{ij} \in \text{Input}(c_i)$ **do**
4: **if** $\exists\, c_{mn}, c_{mo} \mid c_{mn} \in \text{Output}(c_m) \wedge c_{mo} \in \text{Input}(c_m) \wedge \langle c_{mn}, c_{ij} \rangle \in L \wedge \langle c_{mo}, c_{mn} \rangle \in \sigma$ **then**
5: $\text{prop}(c_{mo}) \oplus \text{prop}'(c_{ij}) \mid \text{prop}'(c_{ij}) \leftarrow \text{prop}(c_{mn}) - \text{prop}(c_{ij})$
6: **end if**
7: **end for**
8: **end for**
9: **end procedure**

component inputs during back propagation (Line 3). For each input parameter c_{ij} of each component c_i, the algorithm checks if it is connected to the output parameter c_{mn} of another component c_m (Line 4). If so, and c_{mn} is transitive dependent on an input c_{mo}, then the semantics are propagated to c_{mo} (Line 5). Only semantics which have not already been satisfied by c_{mn} are propagated backwards. The next section talks about the other step that is necessary to generate suggestions for users; component generalisation.

4.3 Component Generalisation

The components in a workflow are generalised to abstract components that represent the functions they perform. This is useful when identifying and extracting logical functional units from workflows. In addition, these generalised units are useful when the system is attempting to intelligently infer the kind of workflow a user is attempting to build. For a detailed discussion of the approach adopted to generalise components, the reader is directed to Section 3.3.8.

The algorithm that implements the approach discussed in Section 3.3.8 is shown in Algorithm 3. G is the graph representing the converted workflow that is to be generalised. The ontology containing the taxonomy of the nodes is O. $N(G)$ is a predicate that returns all the nodes in graph G while n_i and n_{i+1} are the i^{th} and $i + 1^{th}$ nodes respectively. $l(n_i)$ gives the label of the i^{th} node. This label corresponds to the taxonomic category to which this node belongs. The *Generalise_All(G)* operation generalises all the nodes in graph G to the most general term in the ontological taxonomy. The *Specialise(n)* operation, on the other hand replaces the label of the node n with the next specific label in the taxonomy. The algorithm begins by generalising all the nodes in the graph to the most general term in the taxonomy (Line 2). In the main loop, it checks all adjacent node to determine whether they have the same label (Lines 4 and 5). If they do, each of them is specialised to the next specific label in the taxonomy (Lines 6 and 7). The loop ends when all nodes in the taxonomy have been specialised such that no adjacent

Algorithm 3 Component generalisation algorithm.

Inputs:
G: The input graph to be generalised.
O: Ontology describing algorithm taxonomy.

```
1: procedure GENERALISE(G, O)
2:     GENERALISE_ALL(G)
3:     loop                                  ▷ loop over the workflow until all nodes have been generalised
4:         for all n_i, n_{i+1} ∈ N(G) do
5:             if l(n_i) = l(n_{i+1}) then
6:                 SPECIALISE(n_i)
7:                 SPECIALISE(n_{i+1})
8:             end if
9:         end for
10:    end loop
11: end procedure
```

nodes have the same label. The result is an abstract workflow that still retains useful information. Having covered both preprocessing steps required to generate suggestions, the actual algorithm that compiles and sorts the suggestions is discussed in the next section.

4.4 Generating Suggestions

In order to generate suggestions for a partial workflow, it must undergo a series of processing steps. The Component Suggestion Request Handler component is responsible for this task in conjunction with a number of other components. It is discussed in Section 3.3.4. The algorithm that performs this processing and generates suggestions is discussed in Algorithm 4. The inputs to the algorithm are the partial workflow W_p and the domain ontology O containing semantic information about the workflow components. The algorithm consists of two procedures. It starts with the *GenerateSuggestions* procedure. This procedure begins by converting the partial workflow consisting only of concrete components to a graph representing the workflow W_g. This is done by invoking the Workflow to graph conversion component (cp. Section 3.3.7). Once this is complete, the various components in the workflow graph are generalised (Line 3). Some components perform different functions depending on how they are configured. Since HyDRA generalises components based on their functions, a mechanism to deal with such multi-function components is required. In this case HyDRA takes into account additional semantics such as the input and output datatypes of the components to determine which function they are performing in the workflow being generalised. Once generalisation is complete, the generalised graph now contains components which are abstract. As discussed in Section 3.3.8, this allows the system to attempt to intelligently infer what kind of workflow a user is attempting to build. In the third step, the system searches the mined generic patterns stored in the ontology for matching patterns (Line 4). Any matching patterns found are added to a list of suggestion candidates. These candidate are then specialised using the *Specialise* operation. This operation takes the abstract components in the generalised workflow graph and replaces them with concrete components.

Having acquired suggestion candidates from the patterns, the algorithm then attempts to find matching components based on semantics (Line 9). Before that is possible, the workflow has to be semantically analysed to determine what kind of components are compatible with the ones in the partial workflow (Line 8). $C(W_p)$ and $L(W_p)$ are the sets of components and links within the partial workflow W_p. $\sigma(O)$ denotes the set of transitive relationships specified in ontology O (cp. Section 4.2).

Algorithm 4 Suggestion generation algorithm.

Inputs:

W_p	:	Partial workflow that is to be completed.
G	:	Set of original workflow graphs stored in the workflow repository.
O	:	Domain ontology containing semantic descriptions and taxonomic classifications of workflow components.
G_g	:	Abstract partial workflow.
G_{ref}	:	Original specialised partial workflow graph.

```
1:  procedure GenerateSuggestions(W_p, G, O)
2:      W_g ← ConvertToGraph(W_p)
3:      W'_g ← Generalise(W_g, O)                          ▷ see Algorithm 3
4:      C ← SearchPatterns(W'_g, O)                        ▷ get candidates patterns
5:      for all c ∈ C do
6:          Specialise(c, W_g, G)
7:      end for
8:      Analyse(C(W_p), L(W_p), σ(O))                      ▷ see Section 4.2
9:      C ← FindMatchingComponents(W_p,O)                  ▷ append suggestions to candidates
10:     Sort(C)
11:     return C
12: end procedure

13: procedure SearchPatterns(W_g, O)
14:     for p ∈ P(O) do                                    ▷ iterate over every pattern in the ontology
15:         if W_g ⊆ p then                                ▷ if partial workflow is a subset of the pattern
16:             C ← p                                      ▷ add to candidates
17:         end if
18:     end for
19:     return C
20: end procedure

21: procedure Specialise(G_g, G_ref, G)
22:     for all g ∈ G do
23:         return g|g' ⊆ g ∧ g' ⊑ G_g ∧ G_ref ⊆ g'
24:     end for
25: end procedure
```

After matching components have been found, the algorithm searches for matching patterns in the ontology using the same principle in Line 9. Performing semantic analysis on a pattern allows the system to more accurately determine what kind of patterns can be suggested for the current partial workflow. Finally, the accumulated candidates are sorted and then returned to the workflow composition tool being used (Lines 10 and 11). In order to find matching patterns the algorithm uses the *SearchPatterns* procedure. This procedure iterates over all of the patterns stored in the ontology. Every pattern of which the partial workflow graph is a subset is a candidate for suggestions. These candidates are stored in a list and returned to the main calling procedure. Since the patterns as well as the partial workflow graph are generalised, this is equivalent to inferring what kind of workflow a user is attempting to build.

The *Specialise* procedure takes the abstract partial workflow G_g as input along with the original concrete partial workflow G_{ref}. In addition, it also takes the set of original concrete workflow graphs G from the workflow repository as input. It then searches the set G for a subgraph that satisfies certain criteria. This subgraph must be a supergraph of the partial workflow graph G_{ref} being considered. It must also be a specialisation of the abstract workflow graph G_g provided as input to the algorithm. This criterion is introduced because by definition each abstract workflow may have many valid concrete workflows. However, concrete workflows that do not contain the original partial workflow being considered are of no interest to the user building the workflow. Using this criterion ensures that the subgraph returned contains the original partial workflow graph. This concludes the discussion of the

first part of the chapter related to the various algorithms. The next section talks about the domain ontology and its relevance to suggestion generation.

4.5 Domain Ontology

This section talks about the domain ontology used in this research. The ontology is used to perform semantic reasoning about the various workflow components as well as store frequent patterns as functional units. More specifically, during suggestion generation, it is consulted by various components to perform the following functions:

1) Perform semantic analysis of workflows (complete or partial). This involves propagating semantics across workflow components. Transitive dependencies are utilised for this purpose.
2) Match partial workflows to patterns in order to infer what a user is attempting to do.
3) Identify new components using their semantic properties.
4) Determine semantic compatibility between two components.
5) Specify the taxonomy classification of components required for workflow generalisation.

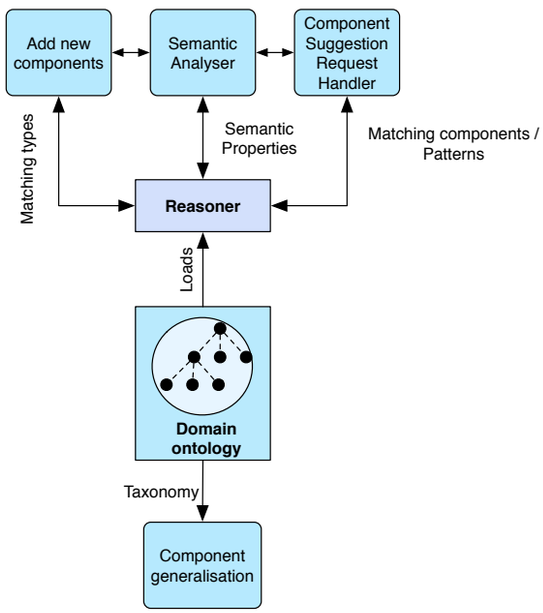


Figure 4.7: Interaction of components with domain ontology.

All of the components that interact with the domain ontology are shown in Figure 3.2. A simplified diagram is shown in Figure 4.7. Each of the functions mentioned previously is covered in the following sections. In order to perform these functions, knowledge about the workflow components and their inter-relationships must be represented in the ontology. Although this section focuses on discussing the ontology implemented as part of this research, the ontology in itself is not the ultimate goal. By discussing the specifics of the ontology this section attempts to illustrate how knowledge representation tools like DL can be used in workflow recommender systems. It can be realised in any ontology modelling language as long as the required features are present.

4.5.1 Taxonomic Classification

One of the functions this framework performs is to generalise the components in a workflow in order to identify functional units. For the generalisation process to work, the components have to be classified in a hierarchical taxonomy. Several criteria can be used to classify the components such as function, package (part of the same software suite), component author, domain etc. However, since the goal here is to identify functional patterns, a taxonomy based on the functions of the components is required. Therefore, this thesis uses an ontology that classifies components according to function. The process of developing a taxonomy based on the functions of workflow components involves the following steps:

1. Perusal of relevant literature and documentation by domain experts to determine the functional classification of the workflow component.
2. Mapping of the functional classification of the component to ontology concepts.

For example, consider the FLIRT component introduced earlier. From the relevant documentation, it was identified that it is an affine registration component. This was mapped to the eponymous ontology concept *AffineRegistrationComponent* which in turn is a *RegistrationComponent*. The resultant taxonomy is shown in Figure 4.3. Following this process, the prototype taxonomy shown in Figure 4.8 has been developed. This taxonomy is an extension of OntoNeuroBase [5]. Since it is a prototype ontology, it is not domain complete. It covers only the dataset that is being used to evaluate this research. For example, another workflow component, also part of FSL, is the *Brain Extraction Tool (BET)*. This component takes an MRI as input and delineates various parts of the brain, a process called *brain segmentation*. To represent this component in the taxonomy, a concept called *BrainSegmentationComponent* was created which is itself a *SegmentationComponent* since other types of segmentations are also possible. In this manner, the prototype taxonomy was incrementally developed.

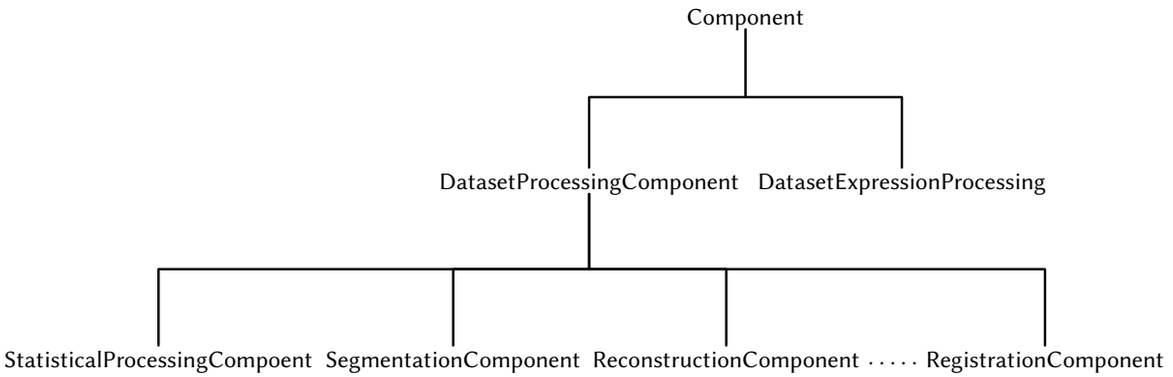


Figure 4.8: Workflow component taxonomy based on function.

During the development of this taxonomy, the OntoClean [6] methodology was followed to justify the design choices made. This methodology and its application are discussed in more detail in Chapter 6. In addition to components, the ontology also contains a taxonomy of the datasets that components take as input and produce as output. These datasets are categorised according to the type of components they are inputs or outputs for. This taxonomy helps to classify components that are new and are not known to the ontology. The methodology for this is discussed in Section 4.5.5. The taxonomy is shown in Figure 4.9. This taxonomy is useful for:

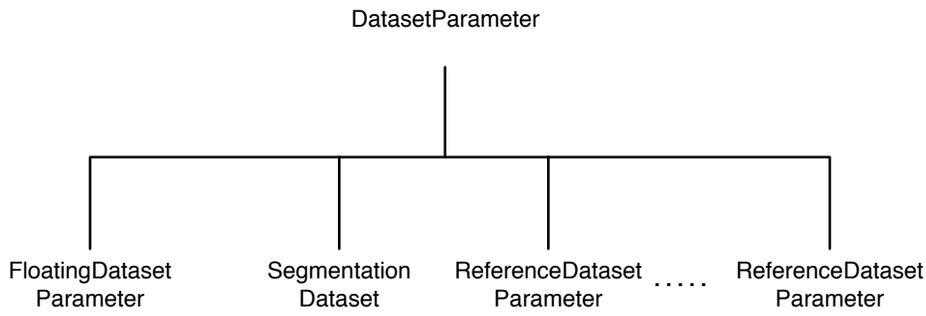


Figure 4.9: Datasets classified according the components that consume or produce them.

1) Determining semantic compatibility between components based on inputs and outputs.
2) Categorising unknown components.

In order to perform these functions, the inputs and outputs of components must first be represented in the ontology. The following section discusses how this is achieved.

4.5.2 Representing Inputs and Outputs

Workflow components have input and output parameters that are primarily of two types; a) *literals* (strings, integers, floats, etc.), and b) *files* (datasets) . Input literals often represent configuration parameters for components and the algorithms they represent. They affect the behaviour of the components and sometimes determine what function they actually perform. This is true for components that can be configured to perform multiple functions. Input files constitute the actual data that is to be processed by the component. On the other hand output literals often represent statistical information about the data that is provided as input to the component. This statistical information is calculated by the component and output for further consumption. Output files usually constitute the actual data that is produced by the component after processing the input data. In neuroimaging, files usually represent the MRI scans that are to be processed by the workflow to extract meaningful information for users (in this case neuroscientists).

A scientific workflow is a sequence of components. It is composed by attaching the output of one component to the input of another. Usually, it is the file inputs and outputs of the components that provide the link between components. Literal inputs and outputs are usually not connected to any other component. This is so because the actual data is constituted by the files that represent the flow of data through the workflow. Since the file parameters form the links between components, a workflow recommender system that helps users build workflows must focus on them. Therefore, for the purposes of this research, only file parameters are considered. In order to support reasoning about the inputs and outputs of components, they must be represented in the ontology as concepts. Since, in the context of workflows, files only make sense when they are thought of as inputs to, or products of components, they are encoded as such in the ontology. The taxonomy shown in Figure 4.9 categorises various files (datasets) using semantics derived from the components they are consumed or produced by.

For example, the FLIRT component takes a *segmented* MRI as input and produces a *registered* image as output. These relationships can be modelled in the ontology by defining two roles *has-for-output* and *has-for-input*. The fact that a *DatasetProcessingComponent* has an input *DatasetParameter* and an

output *DatasetParameter* can be modelled using the following axioms:

$$\exists \text{has-for-input} \,.\, \text{DatasetProcessingComponent} \sqsubseteq \text{DatasetParameter}$$
$$\exists \text{has-for-output} \,.\, \text{DatasetProcessingComponent} \sqsubseteq \text{DatasetParameter}$$

Furthermore, a *RegistrationProcessingComponent* is a *DatasetProcessingComponent* that takes a *SegmentationParameter* as input and produces a *RegistrationParameter* as output. A *SegmentationParameter* is defined as a dataset that has been segmented while a *RegistrationParameter* is defined as a dataset that has been registered. In addition to this, a registration component also takes a *ReferenceDataset* as input to which it aligns the input dataset. These relations can be modelled in the following way:

$$\exists \text{has-for-input} \,.\, \text{RegistrationComponent} \sqsubseteq \text{SegmentationDataset}$$
$$\exists \text{has-for-input} \,.\, \text{RegistrationComponent} \sqsubseteq \text{ReferenceDataset}$$
$$\exists \text{has-for-output} \,.\, \text{RegistrationComponent} \sqsubseteq \text{RegistrationDataset}$$

In addition to having some processing applied on them by workflow components, datasets are also stored in a particular file format. Since the file format pertains to the physical instance of the dataset, it is modelled using the concept *DatasetExpressionFormat*. The role *has-expression-format* relates a *DatasetParameter* to a *DatasetExpressionFormat*:

$$\exists \text{has-expression-format} \,.\, \text{DatasetParameter} \sqsubseteq \text{DatasetExpressionFormat}$$

Given all of these axioms combined, the simple assertion:

$$\text{RegistrationComponent}(\text{FLIRT})$$

would allow the system to determine that *FLIRT* is a registration component that takes as input a reference and a segmented parameter. It produces a registered dataset as a consequence. Thus far, this is more information than is generally stored in a workflow management system about any component. Usually, this information is provided in a free-form text description and is only human-readable. Modelling this information in an ontology makes it machine-readable. To specify semantics of the inputs and outputs more completely, they must be instantiated within the ontology. As a convention, for any workflow component c_i, c_{ij} will be used to refer to its inputs and c_{ik} to its outputs. Assuming $c_1 = \text{FLIRT}$, it can be modelled in the ontology as:

$$\begin{aligned}
&\text{RegistrationComponent}(c_1)\\
&\text{SegmentationDataset}(c_{11})\\
&\text{ReferenceDataset}(c_{12})\\
&\text{RegistrationDataset}(c_{13})\\
&\text{has-for-input}(\text{FLIRT}, c_{11})\\
&\text{has-for-input}(\text{FLIRT}, c_{12})\\
&\text{has-for-output}(\text{FLIRT}, c_{13})
\end{aligned} \tag{4.1}$$

Moreover, *FLIRT* takes its inputs and produces outputs in the *IMG* format. This can be asserted in the following manner:

$$\text{DatasetExpressionFormat}(\text{IMG})$$
$$\text{has-expression-format}(c_{11}, \text{IMG})$$
$$\text{has-expression-format}(c_{12}, \text{IMG})$$
$$\text{has-expression-format}(c_{13}, \text{IMG})$$

Making the input/output parameters explicit like this instead of simply treating them as links between components allows the system to reason about them. This helps the system when identifying unknown components as well as for finding matching components. Both these aspects are covered in Sections 4.5.5 and 4.5.6. In the next section, another important element of the ontology is discussed; transitive dependencies. These are required for performing semantic analysis of a workflow as described in Section 3.3.3.

4.5.3 Representing Transitive Dependencies

A workflow can be thought of as an assembly line. Parts are added to a product in stages until a finished product is achieved. Similarly, in a workflow, data passes through the workflow in stages, being transformed along the way until some meaningful results are achieved. While there are some components that consume data and produce completely new data, there are others that only transform the input data in some way. For example, the *MRI SegStats* component [7] takes a segmented dataset as input and calculates various statistics for it such as the volume of each segment. On the other hand, there are components like *FLIRT* that transform an image, but retain its other properties. It performs linear registration of an input segmented image to a reference image. The output image is also segmented since the input image is segmented. Due to this property of components, sometimes semantic compatibility is not easy to determine. Such an example is discussed in Section 3.3.3 and expanded in Section 4.2. The example shows that sometimes it is not enough to just consider the semantics of the component under immediate consideration. In some cases semantics can be carried across components, requiring a more sophisticated analysis mechanism.

Since semantics cannot be carried across all kinds of components (e.g. *MRI Segstats*), some mechanism is required to discern whether semantic propagation is possible or not. One way to do this automatically would be to analyse the source code of the algorithm behind the component. However, this is not a feasible approach since the source code is generally not available. Moreover, often components have several inputs and several outputs, e.g. *FLIRT*. It is also not generally possible to discern which inputs transfer their semantics to which outputs. Therefore, a semi-automated methodology has been developed to allow the system to perform semantic analysis. This research defines a *transitive dependency* as a relationship between the input and output of a component where the former transfers its semantic to the latter. These dependencies are encoded into the ontology by domain experts. They are later used by the semantic analysis algorithm to propagate semantics across components (cp. Section 4.2).

Transitive dependencies can be captured in the ontology via roles. In the prototype ontology used

in this research, two such roles have been defined; *is-transitive-to* and *is-transitive-from*. Formally:

$$\exists \text{ is-transitive-to}.\, \text{DatasetParameter} \sqsubseteq \text{DatasetParameter}$$
$$\text{is-transitive-from} \equiv \text{is-transitive-to}^{-}$$

These state that *is-transitive-to* is a role that related a *DatasetParameter* to another *DatasetParameter*. Also, *is-transitive-from* is the inverse role of *is-transitive -to*. For the component *FLIRT* as defined in (4.1), there exists a transitive dependency between the output c_{13} and the input c_{11}. This can be captured using the following axiom:

$$\text{is-transitive-to}(c_{11}, c_{13})$$

Consequently, it can be inferred that:

$$\text{is-transitive-from}(c_{13}, c_{11})$$

This relationship is used by Algorithms 1 and 2. For the purposes of the algorithm, the tuple $\langle c_{11}, c_{13} \rangle \in \sigma$ specifies transitive dependency between c_{13} and c_{11}. Now that components can be represented in the ontology, the next section discusses how patterns can be encoded using these components.

4.5.4 Representing Patterns

As discussed in Section 3.2, this framework uses frequent usage patterns to identify functional relationships between components. Once these functional relationships have been identified, the components can be grouped into functional units. Functional units are discussed more in Section 3.3.5. In order to utilise these units for generating suggestions, they must be encoded into the knowledge base.

A functional unit is basically a sub-workflow within a workflow. As such it also consists of a set of components connected to each other via their inputs and outputs. In this case, the functional units consist of generalised components as discussed in Section 3.3.8. Because the generalisation process utilises the functional taxonomy encoded into the ontology, the components comprising the functional units are already present in the ontology. All that remains to be done is to specify the functional relationships between them. The generalised components themselves are represented by concepts in the ontology. To represent the functional unit itself, however, a new concept is required in the ontology. For the purposes of this research, this concept is called a *Pattern*. A functional unit can be thought of as having generalised components as its members. Therefore, it can be modelled in the ontology by specifying that a *Pattern* has *Components* as its members. For this purpose a new role *has-member* is defined. Formally, it can be stated as:

$$\exists \text{ has-member}\,.\, \text{Pattern} \sqsubseteq \text{Component}$$

A component c_1 is *connected* to a component c_2 if c_1 has an output parameter that is an input for

c_2 or vice versa. To capture this relationship, a SWRL [8] rule can be defined in the following manner:

$$\begin{aligned}\text{is-connected}(?c_1, ?c_2) \leftarrow \text{Component}(?c_1) \wedge \text{Component}(?c_2) \\ \wedge \text{has-for-output}(?c_1, ?i) \wedge \text{has-for-input}(?c_2, ?i)\end{aligned} \tag{4.2}$$

Moreover, is-connected is a symmetric relationship, i.e. $\text{is-connected}(c_1, c_2) \implies \text{is-connected}(c_2, c_1)$. Thus:

$$\text{is-connected} \equiv \text{is-connected}^{-}$$

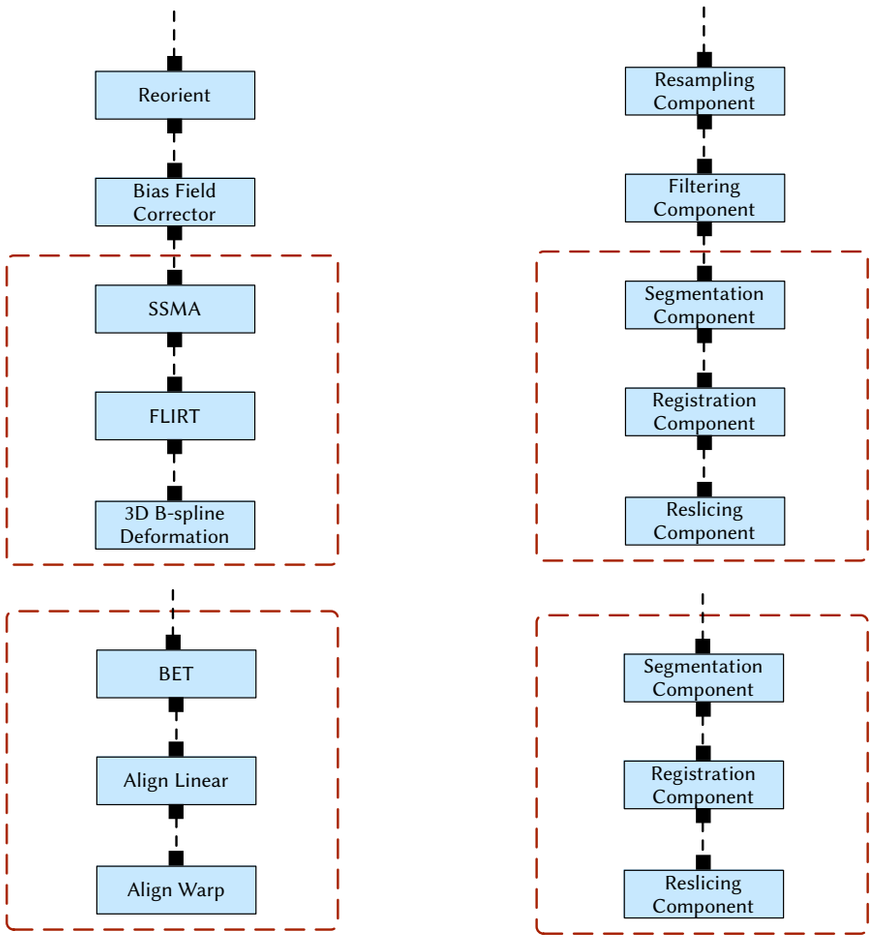
Once it is known that two components are connected, it is easy to determine if a particular component is part of a pattern. A component c_1 is part of a pattern p if c_1 is connected to component c_2 and c_2 is part of p:

$$\begin{aligned}\text{has-member}(?p, ?c_1) \leftarrow \text{Component}(?c_1) \wedge \text{Component}(?c_2) \wedge \text{Pattern}(?p) \\ \wedge \text{is-connected}(?c_1, ?c_2) \wedge \text{has-member}(?p, ?c_2)\end{aligned} \tag{4.3}$$

With the required axioms and rules in place a pattern can now be encoded into the ontology. To understand how this can be done, consider the partial workflows shown in Figure 4.10a. Both snippets are taken from neuroscience workflows and the highlighted snippets essentially perform the same function. They take MRIs as input, segment and register them. Registration itself is a two-step process consisting of registration and reslicing. The workflow components *SSMA*, *FLIRT* and *3D B-spline Deformation* perform these functions in the first workflow respectively. For the second workflow these functions are performed by *BET*, *Align Linear* and *Align Warp*. Figure 4.10b shows the workflow snippets after they have been generalised. As can be seen, once this happens, the system can determine that there is a recurring pattern in the workflows, i.e. the three abstract components *SegmentationComponent*, *RegistrationComponent* and *ReslicingComponent*. These three components can now be grouped into a pattern and added to the ontology. Suppose this pattern is called p_1. The components *RegistrationComponent*, *ReslicingComponent* and *SegmentationComponent* are already present in the ontology as concepts. Therefore, the following axioms can be used to specify that these components are part of the pattern p_1:

$$\text{SegmentationComponent}(sc)$$
$$\text{RegistrationComponent}(rc)$$
$$\text{ReslicingComponent}(r_e c)$$

$$\text{SegmentationDataset}(sd)$$
$$\text{RegistrationDataset}(rd)$$



(a) Original partial workflows.

(b) Generalised partial workflows.

Figure 4.10: Identifying functional units.

$$\text{has-for-output}(sc, sd)$$
$$\text{has-for-input}(rc, sd)$$
$$\text{has-for-output}(rc, rd)$$
$$\text{has-for-input}(r_e c, rd)$$

$$\text{Pattern}(p_1)$$
$$\text{has-member}(p_1, sc)$$

Given these axioms in the ontology, it can be inferred from Rule 4.2 that sc is connected to rc and rc is connected $r_e c$. Since sc is a member of p_1, Rule 4.3 dictates that rc and $r_e c$ are also members of p. Patterns stored in this way are then used by Algorithm 4 in the *SearchPatterns* procedure. Thus far the discussion has only focussed on representing the knowledge. However, it is is only useful if it can help in generating suggestions. The following sections discuss how this knowledge is useful for a workflow recommender system.

4.5.5 Identifying New Components

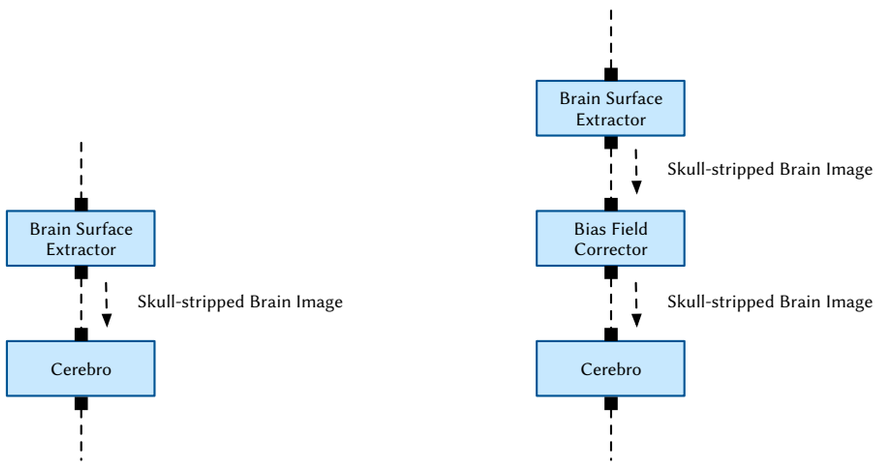
The dynamic nature of the workflow repository means that at any given point in time, there is a significant probability that the knowledge base will be incomplete / outdated. In order to keep the repository up to date, a mechanism to detect the changes and update the ontology accordingly is required. It is assumed that when an algorithm is updated, it is represented as a separate component instead of changing existing components. Therefore, the problem of keeping the ontology up to date can be formulated as detecting new components and inferring their types. For this purpose a novel semi-automated methodology has been defined. The process of identifying new components involves:

1. Propagating semantics across the workflow to estimate what the semantic properties of the unknown component probably are.
2. Pruning extraneous semantic properties.
3. Using the inferred semantic properties of the unknown component to retrieve matching candidate components from the workflow.

The following sections cover each of these steps in turn using examples introduced earlier.

4.5.5.1 The Example

Since in this research components are described using their function, inputs and outputs, if there is enough information available about the inputs and outputs, their function can be inferred. To understand how this is possible, consider the example shown in Figure 4.11, first discussed in Section 3.3.5. Following the conventions introduced earlier, let $c_1 =$ *BrainSurfaceExtractor*, $c_2 =$ *Cerebro* and $c_3 =$ *BiasFieldCorrector*. As before, for each component c_i, c_{ij} refers to its inputs while c_{ik} its outputs. Since each component in this example only has one input and one output each, it can be further specified that c_{i1} will refer to the input of component c_i and c_{12} will refer to its output. Using these conventions, the components can then be encoded into the ontology. The *Brain Surface Extractor* component



(a) Simple example. (b) Complex example.

Figure 4.11: Satisfying semantic requirements.

is a skull-stripping algorithm that extracts brain tissue from an MRI by removing the skull and other

[File Creation Timestamp: 24/02/2016, 14:13]

tissue. It can be described in the ontology using the following axioms:

$$\begin{gathered}\text{SkullStrippingComponent}(c_1)\\ \text{DatasetParameter}(c_{11})\\ \text{SkullStrippedDataset}(c_{12})\\ \text{has-for-input}(c_1, c_{11})\\ \text{has-for-output}(c_1, c_{12})\end{gathered}$$

The *Cerebro* component takes a skull-stripped MRI and delineates the various regions of the brain. The following axioms capture this knowledge:

$$\begin{gathered}\text{BrainSegmentationComponent}(c_2)\\ \text{SkullStrippedDataset}(c_{21})\\ \text{SegmentationComponent}(c_{22})\\ \text{has-for-input}(c_2, c_{21})\\ \text{has-for-output}(c_2, c_{22})\end{gathered} \tag{4.4}$$

The *Bias Field Corrector* is merely a noise-reduction tool. It does not otherwise change the nature of the input MRI. Therefore:

$$\begin{gathered}\text{FilteringComponent}(c_3)\\ \text{DatasetParameter}(c_{31})\\ \text{DatasetParameter}(c_{32})\\ \text{has-for-input}(c_3, c_{31})\\ \text{has-for-output}(c_3, c_{32})\\ \text{is-transitive-to}(c_{31}, c_{32})\end{gathered}$$

Suppose that the *Brain Surface Extractor* is a new component for which semantics have not yet been specified. In the simplest case shown in Figure 4.11a, it is easy for the system to determine *Brain Surface Extractor* is a skull-stripping algorithm since that's what *Cerebro* requires. All that is required is the knowledge that a *SkullStrippingComponent* produces a *SkullStrippedDataset*. This knowledge can be captured using the following axiom:

$$\begin{gathered}\exists\,\text{has-for-output} \,.\, \text{SkullStrippingComponent} \sqsubseteq \text{SkullStrippedDataset}\\ \exists\,\text{has-for-input} \,.\, \text{SkullStrippingComponent} \sqsubseteq \text{DatasetParameter}\end{gathered} \tag{4.5}$$

However, in the more complex example (Figure 4.11b), it is not so trivial to determine how the requirement is being satisfied.

In order to identify the *Brain Surface Extractor*, semantic analysis such as the one performed in Section 4.2 is required. After the semantic analysis algorithm is applied, the requirement for a skull-stripped MRI for *Cerebro* will be propagated backwards to the *Bias Field Corrector*. The system can then determine that *Brain Surface Extractor* is a skull-stripping component. Finally, there is an additional

step that must be performed to ensure that components are identified correctly. Suppose that instead of the *Brain Surface Extractor*, it is the *Bias Field Corrector* that is the unknown component. After semantic propagation, the system now sees that the unknown component is one that takes a skull-stripped image as input and also produces one as output. This is of course, an erroneous conclusion. The correct interpretation is that the skull-stripped property of the image is being transferred from the input to the output, which means that there is a transitive dependency between them. To compensate for this, after semantic propagation is performed, all semantic properties that are common to both the inputs and outputs of a component must be removed. Hence, it must be ensured that given a set of components C in a workflow, $\forall c_i \in C : \text{prop}(c_{ij}) \cap \text{prop}(c_{ik}) = \emptyset$. c_{ij} and c_{ik} are inputs and outputs of the component respectively.

4.5.5.2 Retrieving Candidate Components

Once the semantics are in place, the next step is to query the knowledge base to retrieve candidate components. These are components that have the same semantics as the unknown component. Consider the simplest case shown in Figure 4.11a, i.e. when there are only two components c_i and c_m connected to each other via a link $\langle c_{ij}, c_{mn} \rangle$, c_{ij} and c_{mn} being output and input parameters for c_i and c_m respectively. For this example, it is fairly trivial to find relevant candidates. If c_i is the unknown component, then a query of the following form can retrieve relevant results:

$$\text{get\_candidate}(c_i) = x \mid \exists x, y : \text{Component}(x) \wedge \text{has-for-output}(x, y) \tag{4.6}$$

where

$$\begin{aligned} \text{prop}(c_{ij}) &\sqsubseteq \text{prop}(y) \\ \text{has-expression-format}(c_{ij}) &\sqsubseteq \text{has-expression-format}(y) \end{aligned}$$

The set of candidate components for c_i is the set of components x such that x has an output y. The semantic properties of c_{ij} are subsumed by the semantic properties of c_y. The reason only the output parameter is considered is because only it is relevant. For any component, once semantic analysis is complete and transitive properties have been filtered, any semantic properties available at the output parameter are being likely produced by the component. The same is not necessarily true of the input parameters, however. The semantic properties available at the input may not by required by the component. Therefore, only the output is considered relevant for identifying the component.

To understand how this query gives the desired result, consider the workflow in Figure 4.11a. After the semantic analysis process is complete, $\text{prop}(c_{12}) = \{\text{SkullStrippedDataset}\}$ (from (4.4)). Combining (4.5) and (4.6) it can be seen that *BrainSurfaceExtractor* satisfies the required criteria since it has an output that has the semantic property *SkullStrippedDataset*. Therefore, it is the required candidate component type. However, there is one more case when the process of identifying the component becomes more complex.

Now suppose that *get_candidate* is an operation that retrieves candidate types form the ontology according to (4.6). The system must combine the results from *get_candidate* for all of the output parameters. Therefore, an inclusive method of combining the results from all of the individual outputs

is required. For a particular component c, this can be formally stated as:

$$\forall c_i \in \text{Output}(c) : \mathcal{C} = \bigcup_{j=1}^{i} \text{get\_candidate}(c_j)$$

This equation combines the results for *get_candidate* for all of the output parameters of c via the union operation. The results is a list of candidate types from the ontology for the unknown component based on the semantics of the output parameters that could be inferred from semantic analysis. This list can then be presented to a user to choose from. The user intervention required at this point to identify the correct component type is what makes this methodology semi-automated. The semantic properties obtained after the semantic analysis step may contain some extraneous properties that can be filtered. The principle and methodology for this is discussed in the next section.

4.5.5.3 Pruning Extraneous Semantic Properties

A component may not have just one output. It may have several outputs. It is not possible beforehand to know which outputs are relevant. Therefore, the identification mechanism must take into account all of the output parameters. At this point, it is possible to filter some semantic properties to retrieve more concise candidates. The basic idea is that if one property is more generic than another property, then the more generic property can be filtered. The more specific property will give more targeted results. Formally, it can be stated that for any output parameter c_{ij} and semantic properties $p_1, p_2 \in \text{prop}(c_{ij})$, if $p_1 \sqsubseteq p_2$, then p_2 can be pruned. For example, if $\text{prop}(c_{ij}) = \langle$*DatasetParameter*, *SkullStrippedDatasetParameter*$\rangle$, then *SkullStrippedParameter* $\sqsubseteq$ *DatasetParameter*. Therefore, *DatasetParameter* can be pruned as it does not provide any meaningful information. The same principle can be applied across all the output parameters of the component, i.e. $\exists p_1 \in \text{prop}(c_{ij}), p_2 \in \text{prop}(c_{ik}) : p_1 \sqsubseteq p_2$ for $j \neq k$, then p_2 can be pruned.

The other main function of the ontology in workflow recommender systems is to allow the system to determine matching components. That aspect of the ontology is discussed in the following section.

4.5.6 Determining Matching Components

When suggestions are to be generated, two sources of information are tapped. One are the generalised usage patterns using Algorithm 4. The other source is the semantic information which is used to find components that are semantically compatible with the current one under consideration but may not have been picked up by the usage patterns. Since components are represented using their function, inputs and outputs in the ontology, semantic compatibility must be determined on the basis of these attributes. The function of a component in itself does not provide any useful information to determine semantic compatibility. That information is captured by the usage patterns. Since components interact with each other through their parameters, they must provide the required information. Since component parameters are represented as separate individuals in the ontology, certain semantic assertions can be made about them which can later be reasoned upon.

To determine compatibility based on component parameters, this research utilises the mechanism

used by CAT. In CAT, two components c_i and c_m are compatible iff:

$$\exists c_{ij}, c_{mn} : c_{ij} \in \text{Output}(c_i) \wedge c_{mn} \in \text{Input}(c_m) \tag{4.7}$$

where

$$c_{ij} \sqsubseteq c_{mn}$$

That is to say that c_i has an output and c_m has an input such that the former is subsumed by the latter. This ensures that c_m is a component that can process the output produced by c_i. However, this research stipulates that component compatibility is dependent on one more factor not considered by CAT. All datasets produced and consumed by workflow components have a specific format. A dataset produced in a specific format cannot be consumed by a component that does not accept that format. Therefore, the dataset format is important to consider. In the ontology the concept *DatasetExpressionFormat* captures this format. Therefore, this research modifies (4.7) in the following way:

$$\exists c_{1i}, c_{2j} : c_{1i} \in \text{Output}(c_1) \wedge c_{2j} \in \text{Input}(c_2) \tag{4.8}$$

where

$$c_{ij} \sqsubseteq c_{mn}$$
$$\text{has-expression-format}(c_{ij}) \sqsubseteq \text{has-expression-format}(c_{mn})$$

This states all of what (4.7) stated and introduces an additional condition. c_{ij} has the same format as c_{mn}. Moreover, CAT determines compatibility based only on the immediate semantics. It is does not take into account semantics that can be propagated across components. Contrary to that, this research first performs the semantic analysis discussed in Section 4.2. Consider the workflow in Figure 4.11b. Suppose that *Cerebro* has not yet be added and the system suggests possible completions. At this point, CAT would not consider *Cerebro* a viable candidate since it requires a *SkullStrippedDataset* which *Bias Field Corrector* does not provide. However, since this framework first performs semantic analysis, it will see that there is a *SkullStrippedDataset* available and will thus include *Cerebro* in the list of suggestions. To enable this, (4.8) must include a final modification:

$$\exists c_{ij}, c_{mn} : c_{ij} \in \text{Outputs}(c_i) \wedge c_{mn} \in \text{Inputs}(c_m) \tag{4.9}$$

where

$$\text{prop}(c_{ij}) \sqsubseteq \text{prop}(c_{mn})$$
$$\text{has-expression-format}(c_{ij}) \equiv \text{has-expression-format}(c_{mn})$$

By comparing the semantic properties of the component parameters instead of the parameters themselves, the system utilises the results of the semantic analysis process. This concludes the description of the ontology and how the knowledge encoded in it can be utilised to generate suggestions for users. The next section presents an illustrative example of the interaction of HyDRA with a user followed by conclusions and summarises.

[File Creation Timestamp: 24/02/2016, 14:13]

4.6 Illustrative Example

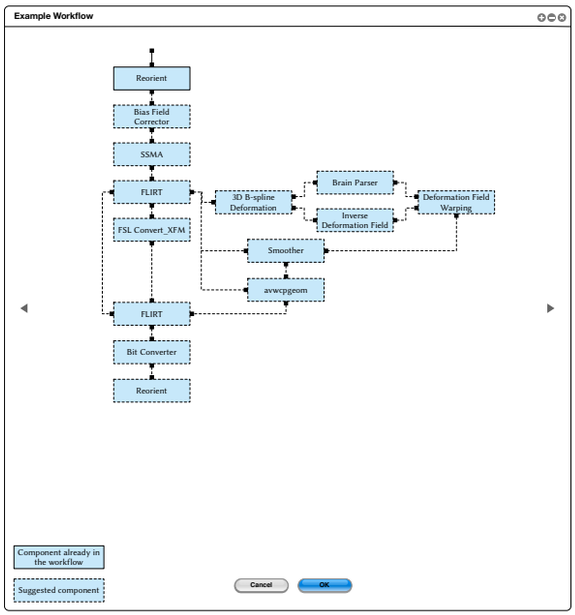
To understand and visualise the interaction of the system with the user, this section demonstrates how HyDRA assists a user in constructing a neuroimaging workflow. Please note this example presents a conceptual visualisation of the way HyDRA would interact with the user. Actual implementations may vary. The example workflow is taken from the LONI repository called "Automated ROI Extraction/Volume Calculation" [9]. As a user adds components to it, the system actively suggests possible completions. Once the user adds the first component "Reorient", HyDRA presents the user with a list of candidates. Selected suggestions are shown in Figures 4.12a to 4.12c. The user can switch between the various suggestions by clicking on the arrows. By clicking 'Ok', the current suggestion can be selected. For this example, the correct candidate consists of the subworkflow in Figure 4.12a. This is because the added component appears in the suggested subworkflow in the repository.

Adding the subworkflow triggers another suggestion generation request. This time, the correct suggestion "MRI Convert" is not in the list, so the user has to manually add the correct component from a list of all components. The next batch of suggestions originate from semantic compatibility between components as they do not appear in any frequent patterns. Once again, selected suggestions are shown in Figures 4.13a to 4.13c and the user can select the correct suggestion by clicking ''Ok". It takes a total of 9 steps to complete this workflow. The completed workflow is shown in Figure 4.14, which is less than the number of components in the workflow. Without HyDRA, the user would require at least as many steps as the number of components in the workflow to complete it.

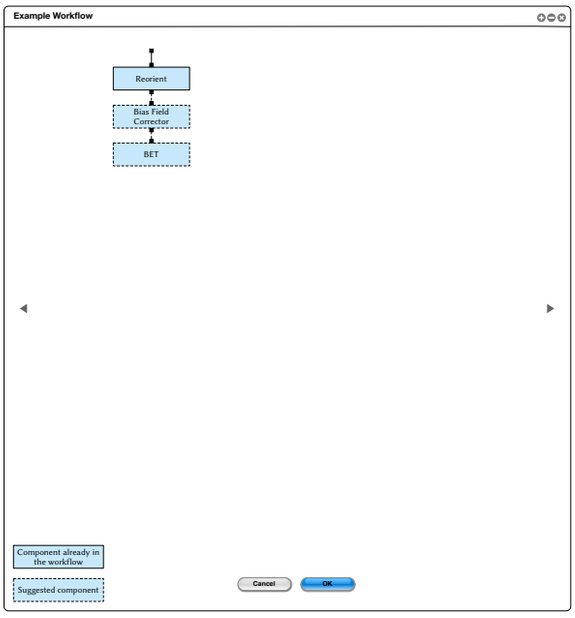
4.7 Summary and Conclusions

This chapter discusses the novel architecture presented in Chapter 3 from a functional perspective. The chapter is subdivided into two main sections. The first section deals with the details of the various components such as algorithms and examples. The second section deals with the details of the domain ontology that is used for generating suggestions. It discusses how knowledge about workflow components can be encoded into the knowledge base and how it can be used to support suggestion generation. The components are represented via a functional taxonomy that classifies them according to function. In addition to this, their output and input parameter types are also encoded. The parameters are represented as datasets that are produced or consumed by the components. Representing them as distinct entities in the knowledge base allows the system to reason about the parameters later on. An illustrative example is also presented to conceptually demonstrate how HyDRA interacts with a user.

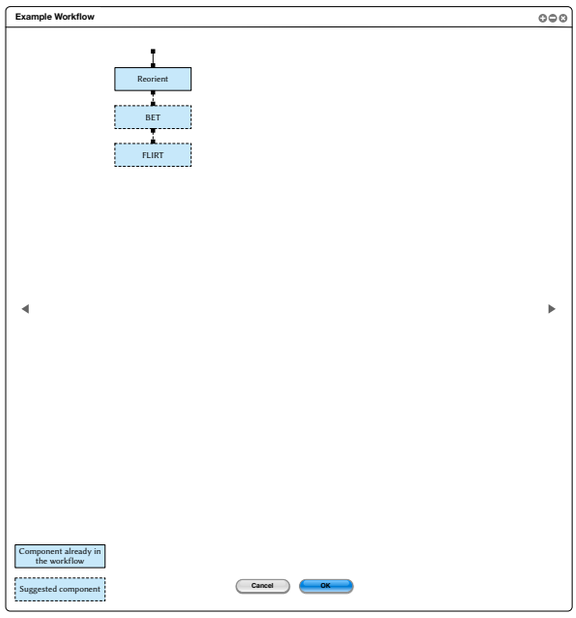
Together with Chapter 3, this chapter shows how semantics and frequent usage patterns can be combined to generate suggestions. The semantics are applied in two phases. In the first phase, the usage patterns are combined with the functional taxonomy to identify generalised patterns in the workflows. These patterns allow the system to attempt and infer what a user is attempting to do to provide contextual recommendations. Shortcomings in existing generalisation approaches are identified and addressed in this research with specific regards to workflows. In the second phase, the semantics of the workflow components are utilised to identify compatibility between components. A novel semantic analysis mechanism is introduced that ensures that the system takes into account sufficient semantic information to produce context-sensitive suggestions. The lack of this semantic analysis mechanism



(a) Suggestion A

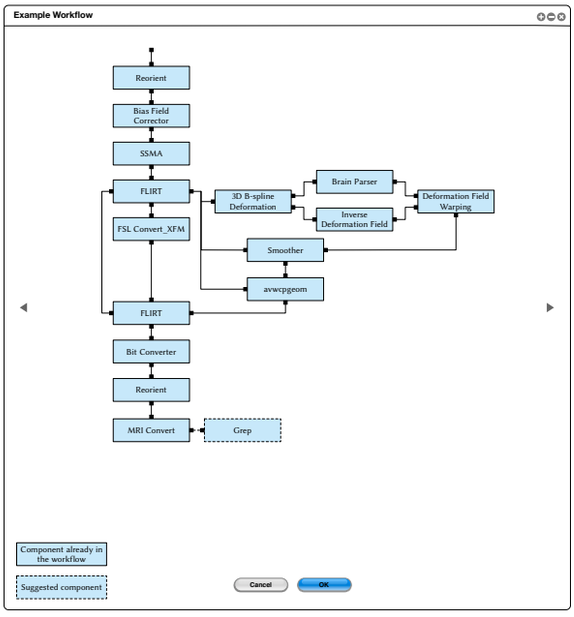


(b) Suggestion B

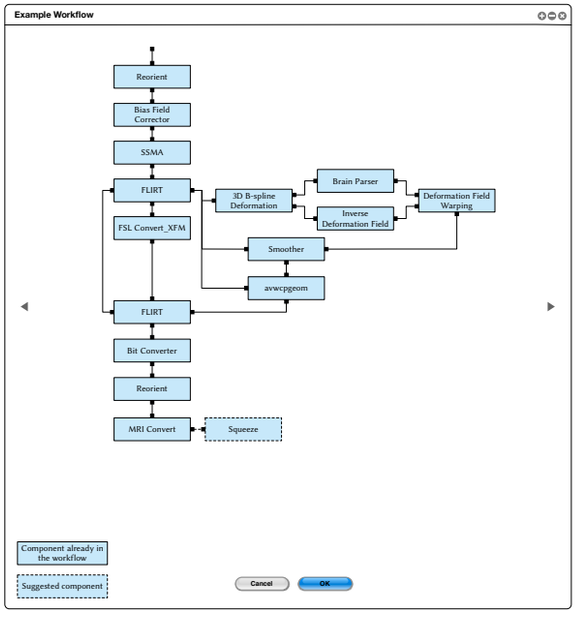


(c) Suggestion C

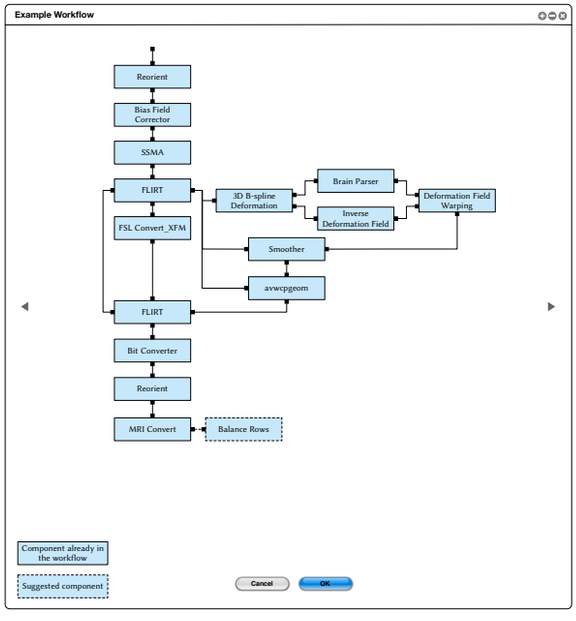
Figure 4.12: Alternative suggestions offered to users while constructing the example workflow. Clicking the arrows allows the user to switch between the suggestions.



(a) Suggestion A



(b) Suggestion B



(c) Suggestion C

Figure 4.13: Alternative suggestions offered to users while constructing the example workflow. Clicking the arrows allows the user to switch between the suggestions.

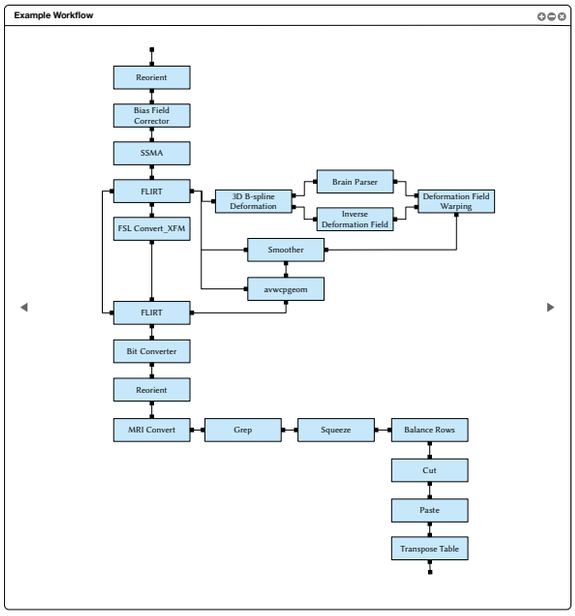


Figure 4.14: Example workflow after 9 steps.

in existing approaches means that the suggestions are generated in the absence of a useful context. The chapter also addresses the question of how the ontology can be kept up to date by introducing a novel semi-automated methodology to detect and identify new components. The drawback to this approach is that it is sensitive to the granularity of semantics specified in the ontology. It is possible in certain cases that there is just not enough semantic information provided for the system to identify the unknown components. Additionally, the approach may not work if multiple components in a workflow are unknown.

Together, Chapters 3 and 4 cooperate to answer Research Question 4:

> *How can workflow component semantics and historical usage patterns be combined to improve the suggestions?*

An architectural framework has been described and analysed that combines both semantics and patterns to generate suggestions. This framework combines the results of querying both sources of knowledge. The semantics of the workflow components are also used to extend the semantic matching of components to entire patterns. This allows the system to determine if a workflow can be completed by suggesting entire patterns instead of single components. In the next chapter the evaluation methodology for this framework is presented.

Bibliography

[1] M. Krötzsch, F. Simancik, and I. Horrocks, "A description logic primer," *arXiv.org*, vol. cs.AI, Jan. 2012.

[2] M. Jenkinson *et al.*, "FSL," *NeuroImage*, vol. 62, no. 2, pp. 782 – 790, 2012, <ce:title>20 {YEARS} {OF} fMRI</ce:title> <ce:subtitle>20 {YEARS} {OF} fMRI</ce:subtitle>. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811911010603

[3] J. Kim, A. Gil, and M. Spraragen, "A knowledge-based approach to interactive workflow composition," in *In Proceedings of the 2004 Workshop on Planning and Scheduling for Web and Grid Services, at the 14th International Conference on Automatic Planning and Scheduling (ICAPS 04)*, 2004.

[4] D. W. Shattuck and R. M. Leahy, "BrainSuite: an automated cortical surface identification tool," *Medical image analysis*, vol. 6, no. 2, pp. 129–142, 2002.

[5] L. Temal *et al.*, "Ontoneurobase: a multi-layered application ontology in neuroimaging," in *Second Workshop: Formal Ontologies Meet Industry (FOMI 2006)*, 2007.

[6] N. Guarino and C. A. Welty, "An overview of OntoClean," in *Handbook on ontologies*. Springer, 2009, pp. 201–220.

[7] https://surfer.nmr.mgh.harvard.edu/fswiki/segstats [Last accessed: 30th June, 2013].

[8] I. Horrocks *et al.*, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member submission*, vol. 21, p. 79, 2004.

[9] "AutoROIExtraction," http://bit.ly/1bJiZ9G.

CHAPTER 5

Evaluation Methodology

5.1 Introduction

Chapter 4 presented algorithms for the various components that constitute HyDRA's framework which must now be verified and evaluated. However, before diving into a discussion of the experimental setup used and results obtained, it is necessary to introduce the evaluation methodology adopted. This chapter serves this purpose. Section 5.2 presents an overview of the methodology with subsequent sections diving into its details. Section 5.7 summarises the chapter.

5.2 Evaluation Methodology

HyDRA's framework (shown in Figure 3.2) comprises several components that embody various sub-processes in the overall suggestion generation process. For this reason, a two-pronged, bottom-up evaluation methodology has been adopted. This methodology consists of both qualitative and quantitative methods. Since there are two main pillars of the framework; patterns and semantics; there are two main branches in the evaluation methodology. The evaluation of each component builds upon the evaluation of other components in the corresponding branch. The methodology along with the evaluation mechanism used for each component is depicted in Figure 5.1. The fundamental component of

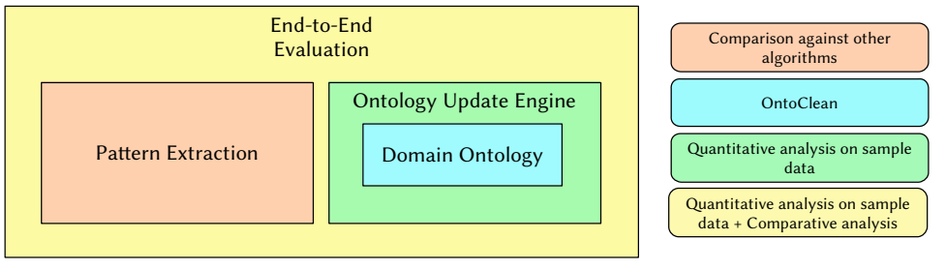


Figure 5.1: Evaluation Methodology

the framework is the domain ontology that contains the semantics used to perform semantic analysis

and generate suggestions. To evaluate it, the OntoClean methodology has been used [1]. The domain Ontology Update Engine (cp. Section 4.5.5) employs the ontology to infer the semantic properties of the unknown component. Therefore, the engine can be successfully evaluated once the ontology itself has been verified. For this purpose, a quantitative analysis method has been used. The engine was tested using sample data (set of sub-workflows). The list of suggestions thus generated was then evaluated using the Mean Reciprocal Rank (MRR) [2]. The MRR is discussed in more detail in Section 5.5. Together these components constitute the semantics-related branch of the evaluation methodology.

The other main branch of the evaluation methodology is the pattern-related branch. Since an existing algorithm has been used to extract patterns, it has been compared to other algorithms that could otherwise have been used. The results are discussed in Section 6.3. Once all of these components have been evaluated, the overall evaluation of the framework can be performed. Since the output of the framework as a whole is also a list of suggestions, an analogous methodology to the domain ontology update engine was applied. The list of suggestions generated by this component have also been analysed using the MRR and also compared to those of another workflow design recommender system (cp. Section 6.5). In the next section, the evaluation of the domain ontology using the OntoClean methodology is discussed.

5.3 The OntoClean Methodology

An ontology consists of structured knowledge about a particular domain. The completeness of an ontology deals with the breadth of knowledge covered by it. The correctness, on the other hand is related to the accuracy of the various design choices made such as the relationships between the various concepts and roles. This research attempts to show how knowledge encoded in a knowledge base such as ontologies can be exploited by workflow recommender systems. Keeping this in mind, we do not endeavour to build a complete ontology. It contains only as much knowledge as is required to cover the available dataset.

The correctness of the ontology, however, is a more complex issue. It consists of two aspects: an objective aspect which pertains to the logical consistency of the ontology, and a more subjective aspect that pertains to the validity of the ontology. The logical consistency can be ascertained automatically with the help of a semantic reasoner such as Pellet [3]. The validity of the ontology, however, is more difficult to test automatically. Generally speaking, there is usually no single way to model an ontology correctly. Depending on the application, a domain may be modelled in several ways, all logically consistent. Due to this ambiguous nature of ontology modelling, they are usually evaluated indirectly [4]. Since the ontology used in this research has been extended from an existing ontology (OntoNeuroBase), the same evaluation methodology that was adopted by its predecessor was used. Therefore, this research adopts the OntoClean methodology for this purpose [1]. The methodology is described below.

The OntoClean methodology consists of applying a set of meta-properties to ontological concepts. These meta-properties impose constraints on whether a subsumption relationship can exist between two concepts or not. OntoClean is designed to be applied throughout the design phase of an ontology. In this sense, the methodology is a qualitative one and helps ontology designers to evaluate different ontology modelling choices and to justify design decisions. It should be noted that OntoClean only

deals with the subjective validity of the ontology. The methodology is shown in Figure 5.2. The first

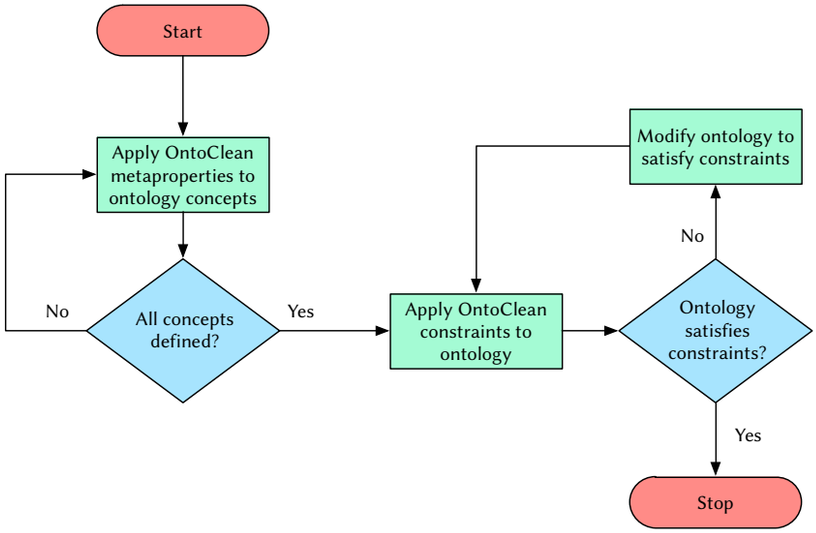


Figure 5.2: Applying the OntoClean methodology.

step is to apply the metaproperties to all the concepts that are part of the ontology. Once all concepts have been defined, the ontology is checked to ensure that the OntoClean constraints are satisfied. If any of the constraints are not satisfied, the ontology is modified appropriately. Once all constraints are satisfied, the process is complete. OntoClean defines the following metaproperties:

1) Identity
2) Rigidity
3) Unity

These metaproperties are discussed in more detail along with appropriate examples in Section 6.2. The constraints imposed by the methodology are also expounded upon later in that section. The next section presents the evaluation methodology adopted for ensuring the correctness of the mined patterns.

5.4 Mining Correct Patterns

In this research pattern mining is performed using the gSpan sub-graph mining algorithm with Closed graph mining enabled. To ensure that the patterns mined are correct and to justify the choice of algorithm, a quantitative evaluation methodology was adopted. The methodology is shown in Figure 5.3. A dataset consisting of a number of workflows was used. In order to ensure correctness, different pattern mining algorithms were applied to this dataset and their outputs compared. In order for the algorithms to be considered interchangeable, their outputs needed to be comparable in two aspects, (a) the number of patterns mined of different sizes, and (b) the number of patterns mined with different frequencies. For example, if one algorithm mines two patterns of size 2 with frequency 4, then all the algorithms must mine those same patterns. The results showed that the outputs from all the algorithms were indeed comparable and are discussed in Section 6.3.

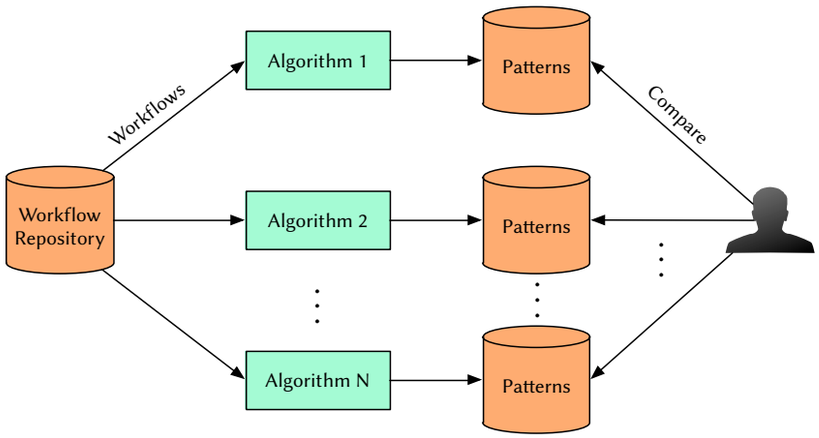


Figure 5.3: Evaluation methodology for ensuring correctness of mined patterns.

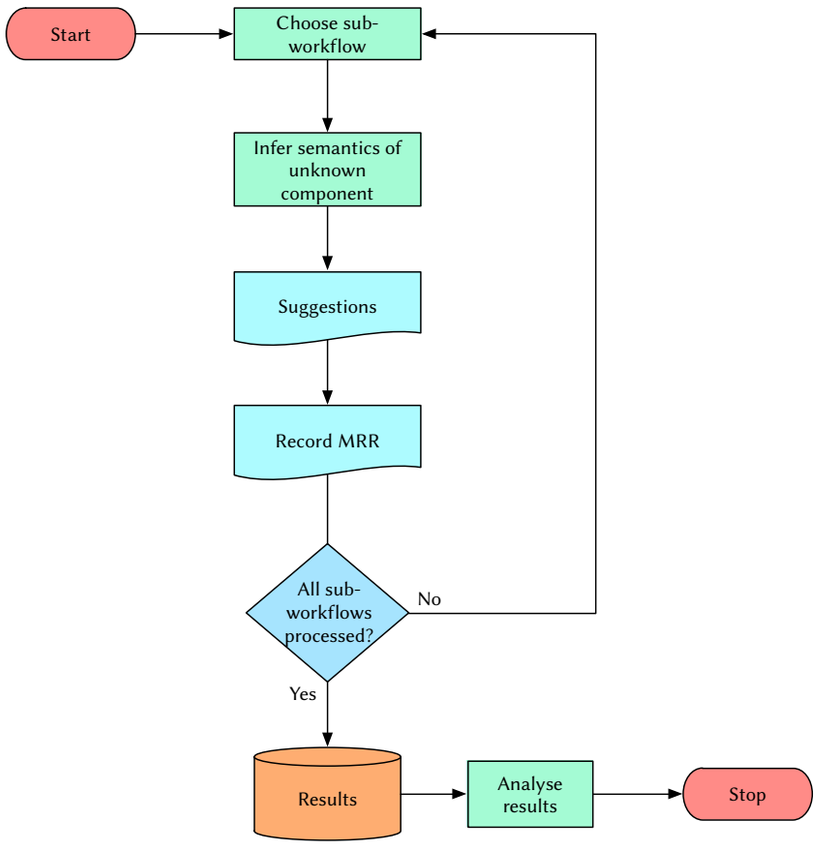


Figure 5.4: Evaluating the semantic inference component.

5.5 Domain Ontology Update Engine Evaluation

The engine is responsible for identifying the type of an unknown component in a workflow by inferring its semantics. To determine if the component performs its function correctly, it is necessary to determine the accuracy of the suggested types. In information theory, there are standard metrics for evaluating the relevance of suggestions for any process that generates them such as Mean Average Precision (MAP) and MRR [2, 5]. MAP is more appropriate for processes that produce suggestions with multiple relevant items. However, for processes that produce suggestions with few relevant items, MRR is suitable [6]. Since in this case there is only one correct suggestion, the MRR is used to quantifiably evaluate the accuracy and usefulness of the suggestions. The measure of MRR in turn gives a measure of the performance of the component. For any suggestion i in a list of suggestions, the MRR of i is given by:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}$$

where $|Q|$ is the number of times the suggestions are generated and rank_i is the rank of i in the list of suggestions. The higher a suggestion is placed in the list, the higher the MRR and the lower the placement, the lower the MRR. The methodology is depicted in Figure 5.4.

A dataset of sub-workflows was chosen as input to the component. For each sub-workflow, this component tried to infer the semantics of the unknown workflow component. The result was a list of suggested workflow component types for which the MRR of the correct suggestion was recorded for every sub-workflow. Once all the sub-workflows had been processed, the results were stored and analysed. The process is further expounded upon in Section 6.4.

5.6 End-to-End Evaluation

In addition to evaluating each of the constituent components, the overall framework was also evaluated. This end-to-end evaluation methodology followed the same pattern as the semantic inference component evaluation methodology. It is shown in Figure 5.5. To evaluate the framework, its performance when constructing a number of workflows was evaluated. After selecting a workflow to construct, components were incrementally added to it. After adding each component, the suggestions generated by the system were evaluated using the MRR of the relevant suggestion. This process was repeated until the workflow was completely constructed. The overall performance of the framework for the workflow was measured by the average MRR for the individual components. Finally, the results were collected and analysed. The experimental setup, along with the results are discussed in Section 6.5.

5.7 Summary

This chapter serves as a prelude to the next chapter in which HyDRA's framework is evaluated. A two-pronged, mixed qualitative and quantitative methodology has been adopted for this purpose. This methodology involves evaluating the different components of the framework separately using methodologies appropriate to those components. These components are the domain ontology, the

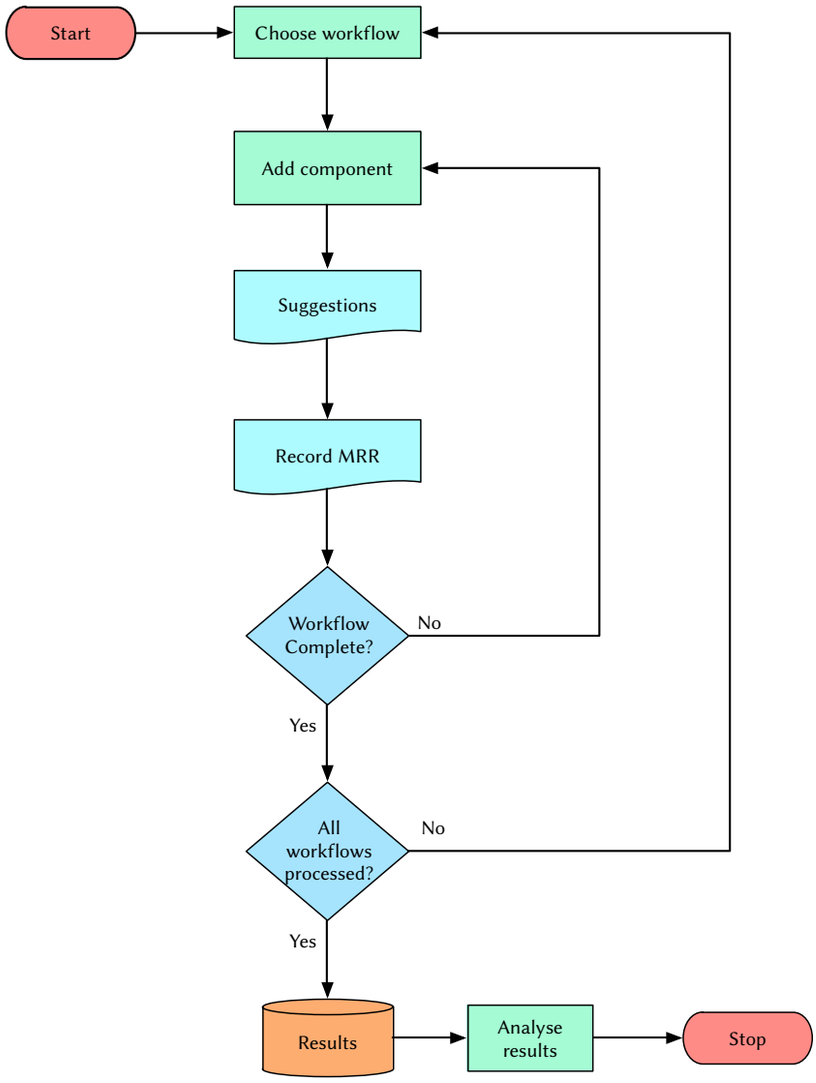


Figure 5.5: End-to-end evaluation methodology.

pattern extraction component and the domain ontology update engine. The semantics-related components; the domain ontology and the domain ontology update engine form one prong of the methodology. The pattern extraction component forms the other prong. In addition, an end-to-end evaluation of the framework is also presented. In the next chapter, the results of the various methodologies are presented and discussed.

Bibliography

[1] N. Guarino and C. A. Welty, "An overview of OntoClean," in *Handbook on ontologies*. Springer, 2009, pp. 201–220.

[2] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008, vol. 1.

[3] E. Sirin *et al.*, "Pellet: A practical OWL-DL reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[4] J. Brank, M. Grobelnik, and D. Mladenić, "A survey of ontology evaluation techniques," in *In In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005)*, 2005.

[5] D. R. Radev *et al.*, "Evaluating web-based question answering systems." in *LREC*. European Language Resources Association, 2002.

[6] N. Craswell, "Mean reciprocal rank," in *Encyclopedia of Database Systems*, L. LIU and M. ÖZSU, Eds. Springer US, 2009, pp. 1703–1703. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9_488

CHAPTER 6

Evaluation and Experimentation

6.1 Introduction

In Chapter 4 the various components of HyDRA's hybrid architecture and their algorithms were discussed. An evaluation of the various components is presented in this chapter and their performance investigated in different scenarios. The various components of the framework that need to be evaluated comprise:

1. The domain ontology.
2. The pattern extraction component.
3. The domain ontology update engine.
4. End-to-end evaluation of the overall system.

The overall evaluation methodology is discussed in Section 5.2. This chapter starts by evaluating the domain ontology by using the OntoClean methodology [1], discussed in Section 6.2. This is followed by an evaluation of the pattern extraction component in Section 6.3. Section 6.4 discusses the input data for the domain ontology update engine as well as the results of the evaluation. Lastly, an overall end-to-end evaluation is presented in Section 6.5 and the chapter is concluded in Section 6.6.

6.2 The OntoClean Methodology

The OntoClean methodology is a set of domain-independent metaproperties that constitute the criteria using which an ontology can be evaluated. These metaproperties help to explicitly evaluate various design choices and justify those made. In addition, it forces ontology modellers to think about why they are modelling the ontology in a particular way and to make their thought process explicit. This is especially important since generally speaking there is no single correct way to model an ontology. The correctness of an ontology is heavily dependent on the application it is intended for. For example, if one is attempting to model the concepts, *time interval* and *time duration*, intuitively one might say that every time interval is a time duration. This would generally be modelled as a subsump-

tion relationship:

$$\text{timeinterval} \sqsubseteq \text{timeduration}$$

One of the consequences of modelling the two concepts in this manner is that any two time intervals $i_1 \neq i_2$ must also be two distinct time durations. They cannot be two distinct time intervals while being the same time duration. However, using the OntoClean methodology, it can be shown that this is not the case. One of the metaproperties used in this methodology is the *identity criteria*. The identity criteria are the characteristics of individuals that allow us to distinguish between them. In essence, these are the criteria that we can use to determine if $i_1 = i_2$ or not. Suppose that in the previous example, we have various time durations like *1 hour*, *2 hours*, *3 hours* etc. We also have certain time intervals like *1:00–2:00*, *2:00–3:00* etc. Now, both the intervals *1:00–2:00* and *2:00–3:00* are the same duration, i.e. *1 hour*. However, they are separate time intervals. Therefore, it is clear that the identity criteria for time interval and time duration are different. According to the OntoClean methodology, it would not be correct to model them as a subsumption relationship. It would be more appropriate to say that every time interval *has* a time duration. This can be captured by defining a new role:

$$\exists\, \text{has-duration} \,.\, \text{timeinterval} \sqsubseteq \text{timeduration}$$

The previous example illustrates how applying the OntoClean methodology can help ontology modellers to evaluate various design choices and identify any logical inconsistencies. The rest of the metaproperties defined by this methodology are discussed in subsequent sections along with examples relevant to the neuroimaging domain.

6.2.1 Identity

As discussed previously, the *identity* criteria help in determining whether two individuals that belong to a particular concept in an ontology are the same or different. It should be noted that identity criteria are different from the membership criteria that allow those individuals to belong to those concepts. With regards to neuroimaging, consider the ontology shown in Figure 6.1. A *DatasetParameter* is an MRI that is produced or consumed by a workflow component. A *SkullStrippedDatasetParameter* is an MRI that is produced by a *SkullStrippingComponent*. It is intuitive to say that a *SkullStripped-DatasetParameter* is also a *DatasetParameter*, i.e.:

$$\text{SkullStrippedDatasetParameter} \sqsubseteq \text{DatasetParameter}$$

But to determine if this choice of modelling is correct, it must pass the identity criteria test that OntoClean recommends. A dataset is characterised by several properties. Amongst them are the patient to whom it belongs and the mechanism through which it was acquired or produced. It can be said that two datasets are the same if they belong to the same patient and the method through which they were acquired or produced is the same. Both of these criteria hold true for a *SkullStrippedDatasetParameter* as well as a *DatasetParameter*. Therefore, it can be said they both carry the same identity criteria and thus they pass the test. This implies that the choice of modelling can be deemed correct or valid. Let us now look at a more complex example of applying this criteria to the neuroimaging domain.

In addition to simple semantics, the ontology in this research also stores frequent usage patterns.

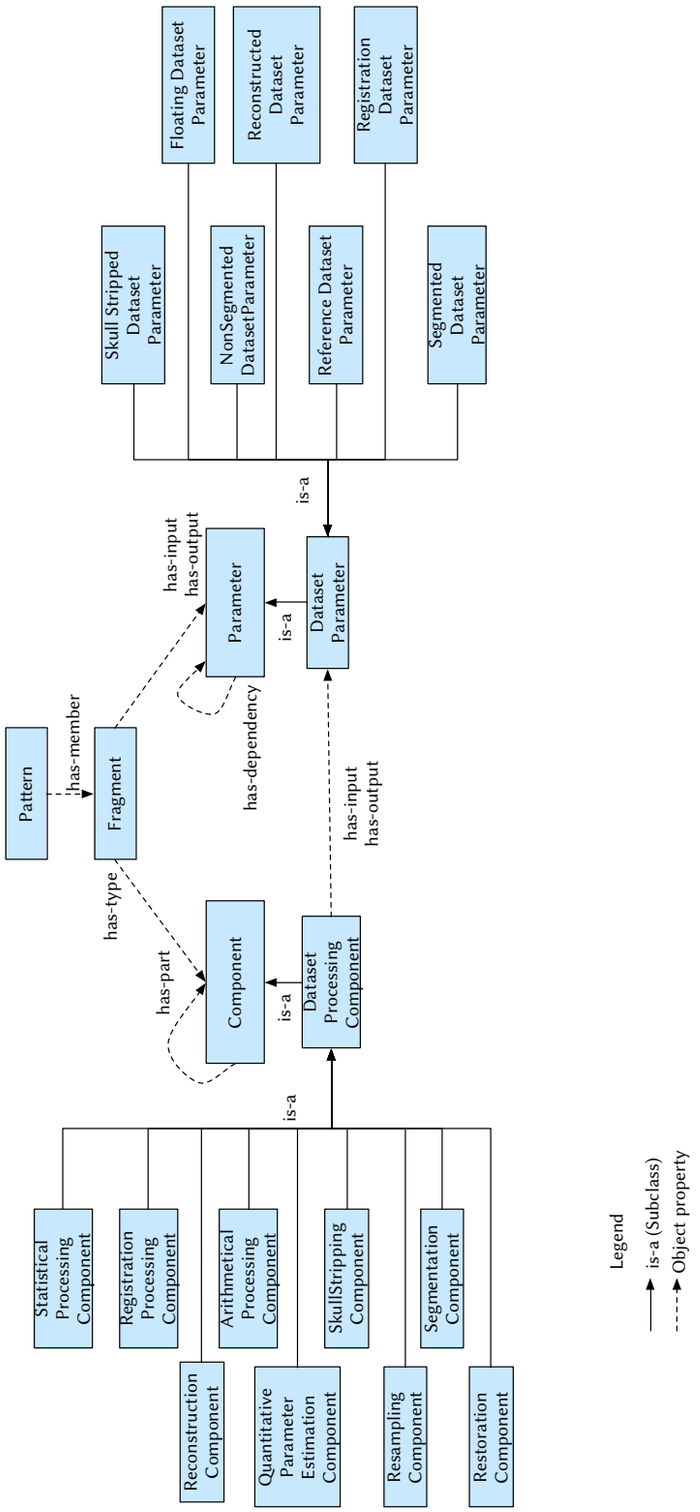


Figure 6.1: Domain ontology.

A pattern is composed of various workflow components linked together. Now workflow components have already been modelled as the concept *Component*. However, when workflow components are part of a pattern, they can be seen as different entities. A workflow component takes an input dataset and produces an output dataset. The dataset produced by one instance of the component when it is part of one pattern will not be the same as the dataset produced by another instance of the same component when it is part of another pattern. This is so because the dataset will have undergone different processes in different patterns. Therefore, using the identity criteria, it can be argued that each workflow component that is part of a pattern *has* a type that is a *Component*. This can be modelled using a role:

$$\exists \text{Fragment. has-type} \sqsubseteq \text{Component}$$

The concept *Fragment* represents all workflow components that are part of some pattern. This way, workflow components that are part of different patterns can have the same *type*, signifying that they perform the same function.

6.2.2 Rigidity

The second metaproperty specified by OntoClean is *rigidity*. Properties that must hold true for every instance in an ontology are called rigid properties. Properties that must hold true for only some of the instances are called semi-rigid properties. Those that must not hold true for all of the instances are called anti-rigid. The rigidity of properties imposes restrictions on the subsumption relationship. More specifically, an anti-rigid property cannot subsume a rigid property. For example, a *Dataset-ProcessingComponent* in the prototype ontology is a workflow component that performs some kind of processing on an MRI. *FLIRT* is an example of such a component. Now, a dataset processing component will always be a dataset processing component. There is no situation where such a component can cease to be one. Therefore, being a dataset processing component is an essential property for such a component, and thus it is rigid. Similarly, a *RegistrationComponent* is a *DatasetProcessingComponent* that performs registration. There is no situation where a registration component stops being one. Therefore, it is also a rigid property. Thus, it can be safely stated that a *DatasetProcessingComponent* can subsume a *RegistrationComponent*. Generally speaking, all of the different types of components are rigid.

Contrary to workflow components, datasets themselves undergo various processing as they pass through the workflow. These processing steps transform them, changing their properties. For example, an unsegmented MRI becomes a segmented MRI after it has been processed by a *SegmentationComponent*. Therefore, all dataset instances might be a particular type of dataset in one situation, and another type of dataset in another situation. Hence, the dataset types are anti-rigid properties since they are non-essential for all datasets. However, a dataset will always remain a dataset. There is no situation where a dataset can become something else. Therefore, the concept *DatasetParameter*, representing a dataset in an ontology, is a rigid property. All dataset types such as *Statistical DatasetParameter* and *SegmentedDatasetParameter* are subsumed by *DatasetParameter*. Since in this case a rigid property is subsuming anti-rigid properties, this is not a violation of the restriction OntoClean imposes on subsumption relationships.

6.2.3 Unity

The third metaproperty used in OntoClean is *Unity*. The unity of an entity is the means through which the parts of the entity are described. The unity criteria of an entity are the relationships that must exist between the parts of the entity. Entities whose parts must be wholes with common unity criteria are said to have unity. Entities whose parts must be whole but with no common criteria are said to have no unity. Lastly, entities whose parts may not necessarily be wholes are said to have anti-unity. Due to their nature, it can be argued that entities with anti-unity cannot subsume entities with unity or no unity. With reference to neuroimaging, workflow components are wholes that have an executable algorithm and inputs and outputs that are themselves wholes. Since the executable algorithms are clearly different entities from their parameters, the components can be said to have no unity. Therefore, the concept *Component* in our ontology has no unity. Any specific type of component, such as *RegistrationComponent* has the same unity criteria. Therefore, it also has no unity. Thus, it can be safely stated that *Component* subsumes *RegistrationComponent*.

Datasets, on the other hand consist of image(s). Images may have different parts that can be identified by segmentation but on the whole it can be said that an image is an indivisible whole. Therefore, a dataset has unity. Similarly, it can be shown that all the different types of datasets also have unity. This fact allows the generic *DatasetParameter* concept to subsume all the various dataset types such as *SegmentedDatasetParameter*.

6.2.4 Discussion

The metaproperties and their semantics have some repercussions for the subsumption relationship. Some of these have already been hinted at. They are expressed more formally here. Given two ontological concepts p and q where q subsumes p, the following constraints must be observed:

1. If q is anti-rigid, then p must be anti-rigid.
2. If q carries an identity criterion, then p must carry the same identity criterion.
3. If q carries a unity criterion, then p must carry the same unity criterion.
4. If q has anti-unity, then p must also have anti-unity.

The application of the OntoClean methodology in essence involves analysing the ontology using the aforementioned metaproperties and ensuring that these constraints are observed. They help to justify the design choices made and also make the semantics of the ontological concepts explicit. Examples of how this methodology can be applied in this research have already been presented. The concepts in the ontology are described exhaustively using these metaproperties in Appendix A. Having described the ontology evaluation methodology, the next section discusses the evaluation of the other major component of HyDRA; the pattern mining component.

6.3 Mining Correct Patterns

The problem of mining frequent patterns in workflows can be formulated as a frequent subgraph mining problem. This is due to the inherent graph-like structure of workflows. Frequent subgraph miners attempt to exhaustively find subgraphs in a graph or set of graphs that occur with a certain frequency. Generally, they start by finding the smallest subgraphs in a graph, and extend them incrementally. The search space in such problems is generally quite large. Therefore, for reasons of time and

space efficiency, only subgraphs that occur frequently are extended further. Those that are not found to be frequent are not extended. Thus, the search algorithm prunes certain parts of the search space intelligently. To determine whether a subgraph is frequent or not, a minimum frequency threshold is used. This threshold often appears as an externally configurable parameter of the search algorithm.

The search mechanism described previously can be thought of as a combinatorial process since it starts with the smallest subgraphs and extends them incrementally. For this reason, the number of subgraphs of size r for a graph of size n can be given by:

$$\text{number of subgraphs} = \binom{n}{r}$$

The total number of subgraphs of all sizes for a particular graph is thus:

$$\text{total number of subgraphs} = \sum_{r=1}^{n} \binom{n}{r}$$

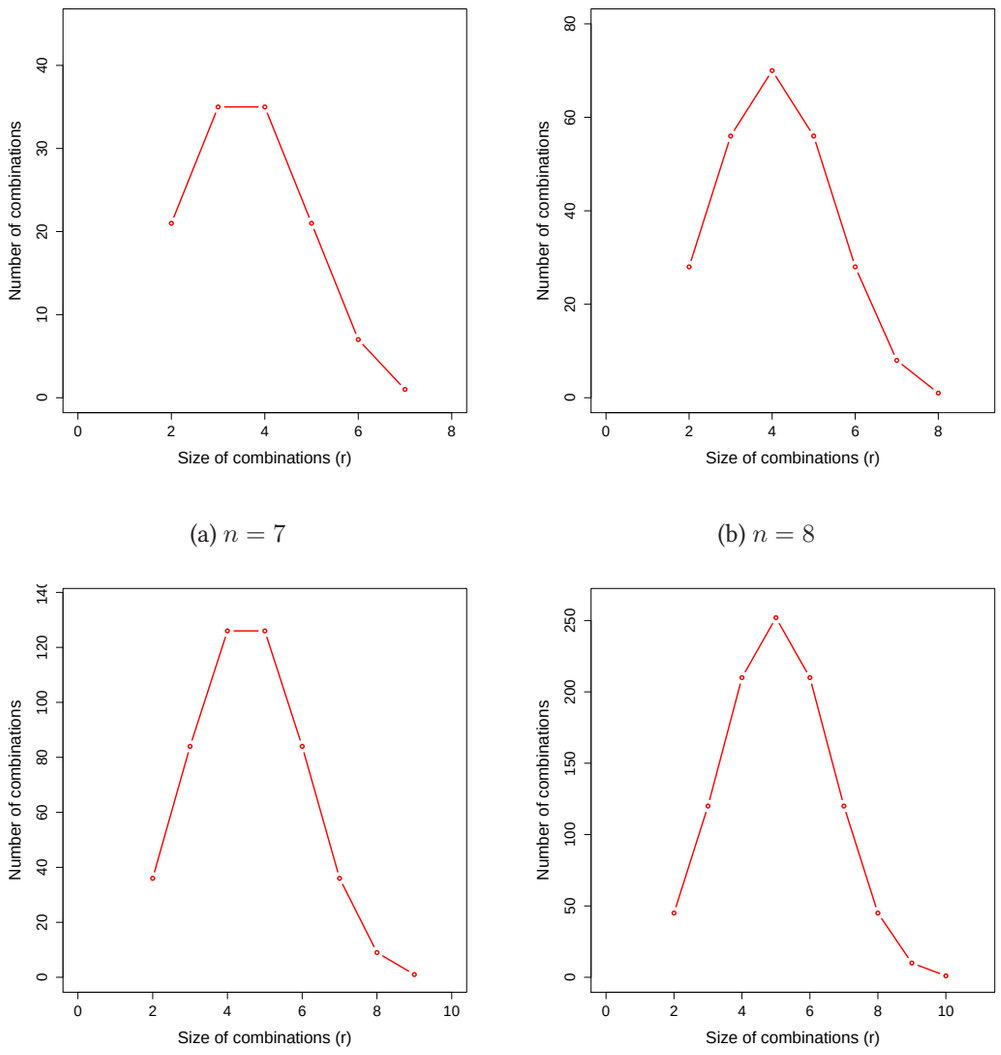
The number of subgraphs of each size follow the patterns shown in Figure 6.2. Figures 6.2a to 6.2d show the maximum number of subgraphs of different sizes that are possible for graphs of sizes 7, 8, 9 and 10. The number of actual frequent subgraphs found will be less than this theoretical limit for each subgraph size. To determine whether the patterns or subgraphs mined by the chosen algorithms are correct, they must satisfy two criteria:

1) Different algorithms applied on the same dataset must produce approximately the same number of subgraphs.
2) The number of subgraphs found must follow the pattern shown in Figure 6.2.

To determine the correctness, various algorithms were applied to a dataset of 73 neuroimaging workflows from the LONI repository [2]. For these tests, the minimum frequency was set to 4 since it is approximately equal to 5% of 73, which is the significance value used in statistical significance tests [3]. The results of the tests are shown in Figure 6.3. As can be seen, for each size, the number of subgraphs follows the patterns shown in Figure 6.2. The total number of subgraphs also matches for the different algorithms. Thus it can be concluded that the patterns mined are correct.

6.4 Domain Ontology Update Engine

This engine is responsible for identifying unknown components based on their inferred semantics. The setup used to evaluate this component is shown in Figure 6.4. A set of sub-workflows, after being semantically analysed, are provided as input to this engine, which generates the list of suggestions. It should be noted that complete workflows could just as easily have been used. However, to facilitate brevity and conciseness when presenting the input dataset and discussing the results, sub-workflows were chosen. These suggestions are evaluated using the MRR of the correct suggestion. This gives a quantitative measure of the performance and correctness of the engine. For the input dataset, five sub-workflow have been chosen. These sub-workflows have been taken from the LONI dataset being used to evaluate this research for their illustrative characteristics pertinent to the performance of this engine. Each sub-workflow helps to demonstrate a particular aspect of the engine or its performance in a particular scenario. The sub-workflows and their details are discussed in the following sections. These are followed by a description of the experimental setup along with results.



(a) $n = 7$ (b) $n = 8$

(c) $n = 9$ (d) $n = 10$

Figure 6.2: Total number of subgraphs of different sizes for different sized graphs.

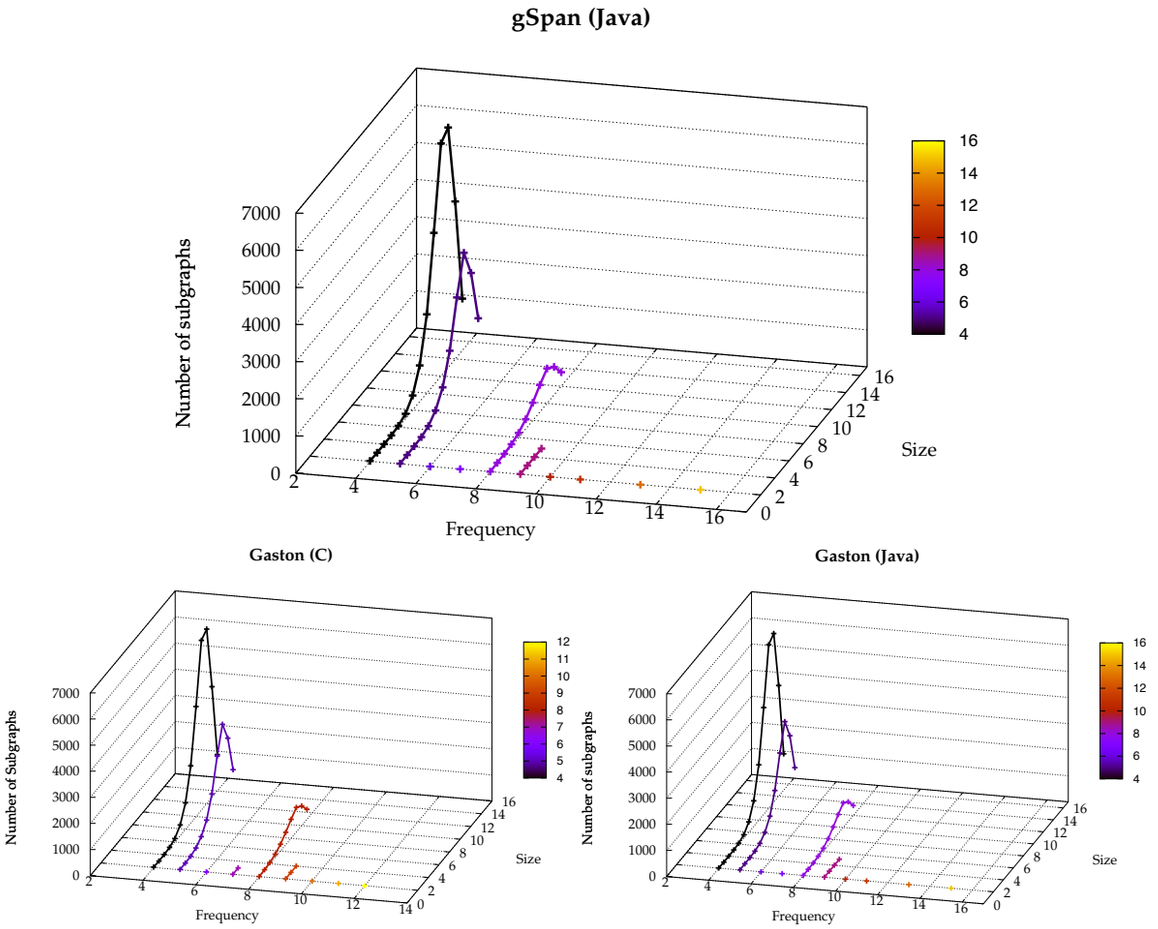


Figure 6.3: Results of frequent subgraph mining tests with minimum frequency = 4.

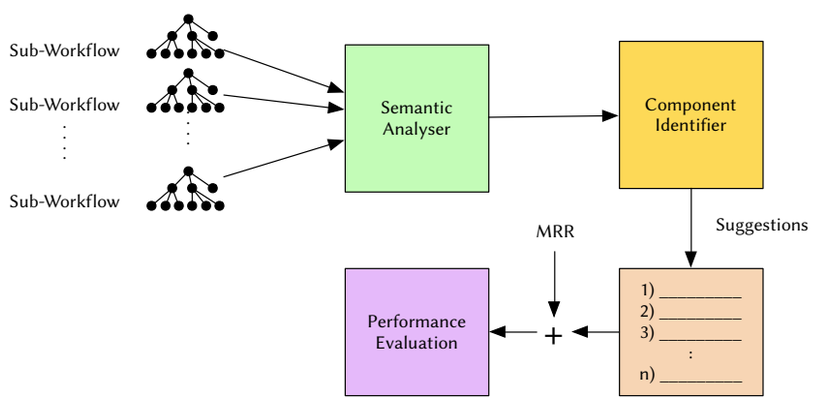


Figure 6.4: Experimental setup used to evaluate the domain ontology update engine.

[File Creation Timestamp: 24/02/2016, 14:13]

6.4.1 Dataset

This section describes the sub-workflows used to evaluate the domain ontology update engine.

6.4.1.1 Sub-Workflow 1 (SW_1)

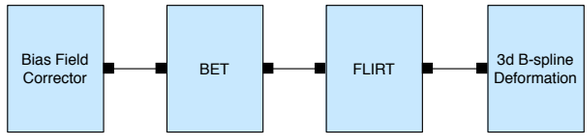


Figure 6.5: Sub-workflow taken from "BrainParser (Hippocampus)"

The sub-workflow shown in Figure 6.5 is taken from the "BrainParser" workflow [4]. This workflow detects the hippocampus in an MRI. It achieves this by first registering the MRI of a patient to a reference MRI. The registered image is then segmented by a *segmentation* algorithm. The image is then warped back into its original form before registration and the segmented result is output. The snippet shown in Figure 6.5 performs the following steps:

- *Bias Field Correction:* takes the MRI and performs noise-reduction to improve signal-to-noise ratio.
- *BET:* this component takes the MRI and removes the surrounding skull-tissue to expose the brain. It is a skull-stripping component.
- *FLIRT:* takes the MRI and performs linear registration according to a reference image.
- *3D B-spline Deformation:* this component is also a registration component. However, contrary to *FLIRT*, it performs non-linear registration of the image with respect to a reference image.

In this snippet, both *FLIRT* and *3D B-spline Deformation* require a skull-stripped image as input since they are both registration components.

6.4.1.2 Sub-Workflow 2 (SW_2)

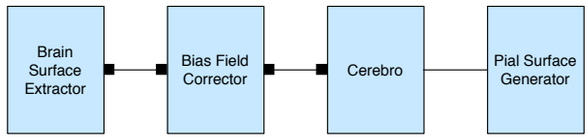


Figure 6.6: Sub-workflow taken from the "Cortical Surface Extraction" workflow.

The sub-workflow shown in Figure 6.6 is taken from the *Cortical Surface Extraction* workflow. This workflow is part of the *BrainSuite* suite of algorithms [5]. This workflow takes an MRI, segments it and separates the two hemispheres of the brain. The snippet shown in the diagram performs the following steps:

1. *Brain Surface Extractor:* takes an MRI and strips the skull to extract the brain.
2. *Bias Field Corrector:* removes noise from an input MRI and homogenises the intensities of the voxels.
3. *Cerebro:* this components segments the brain and labels the cerebellum.
4. *Pial Surface Generator:* this component computes and generates the pial surface of the brain.

In this snippet, the *Pial Surface generator* requires a segmented image while *Cerebro* requires a skull-stripped brain image.

6.4.1.3 Sub-Workflow 3 (SW_3)

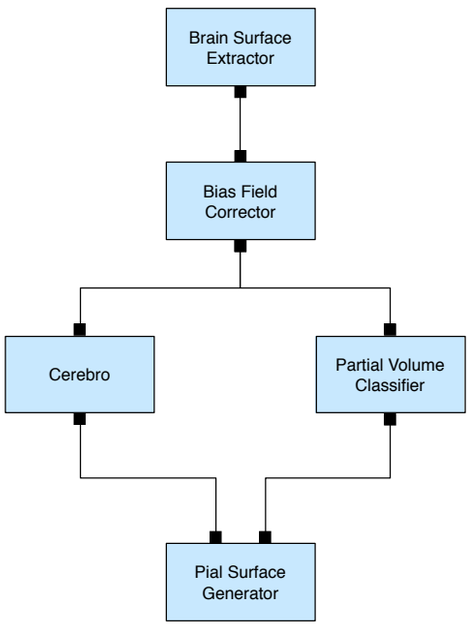


Figure 6.7: Sub-workflow taken from the "Cortical Surface Extraction" workflow.

As with SW_2, SW_3 (Figure 6.7) is also taken from the *Cortical Surface Extraction* workflow. Compared to SW_2, however, SW_3 has two parallel branches of execution. One branch performs the same function as SW_2. The other branch takes the MRI, segments it and extracts the brain cortex. In addition to the brain surface extraction and bias field correction, SW_3 contains the following steps:

1. *Partial Volume Classifier:* segments an input MRI and classifies brain tissue. The input image should be bias-corrected and skull-stripped before it can be segmented by this component.
2. *Cortex:* extracts the cerebral cortex from the segmented brain image.

6.4.1.4 Sub-Workflow 4 (SW_4)

SW_4, shown in Figure 6.8 is taken from the "fMRI Processing Using AIR" workflow [6]. This workflow takes a set of MRIs, registers them and then outputs their mean. It is a rather complex workflow and involves a number of steps to achieve the desired output. The intermediate outputs produced are also saved along the way. SW_4 is the slice of the workflow that takes an MRI, resizes it, registers it and then saves the output. The steps involved in this process are:

- *Resize:* resizes an input MRI to dimensions specified as input.
- *Align Linear:* estimates the transformation matrix required to align (register) an input image to a template image.
- *Reslice:* applies the transformation matrix calculated in the previous step to the input image. After this step the input image matches the coordinate system and proportions of the template

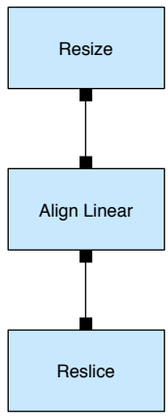


Figure 6.8: Sub-workflow taken from "fMRI Preprocessing Using Air".

image. Together with the previous step, these two steps constitute a registration process between an input MRI and a template MRI.

6.4.1.5 Sub-Workflow 5 (SW_5)

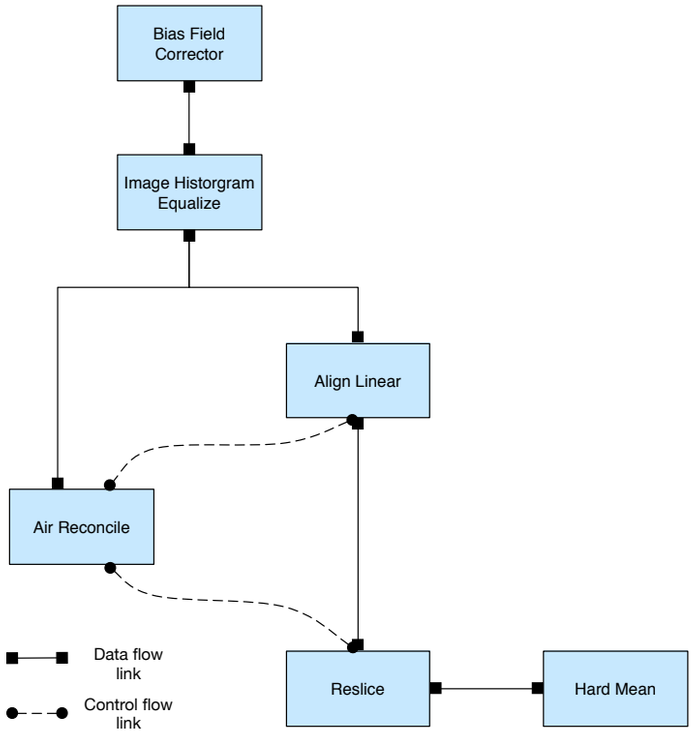


Figure 6.9: Sub-workflow taken from the "Air Reconcile (V2) workflow."

SW_5 is a workflow slice taken from the *Air Reconcile (V2)* workflow, shown in Figure 6.9. It takes a number of MRIs, bias-corrects them and equalises their intensities. It then registers them to each other, calculates their mean and saves the output. It consists of the following steps:

- *Bias Field Corrector:* removes inhomogeneity in the intensity values of the voxels, thus removing errors from the images.

- *Image Histogram Equalise:* this step equalises the intensity values of all the images to ensure that they all are uniform for further processing.
- *Air Reconcile:* this step removes discrepancies between the various .air files.
- *Align Linear:* estimates a transformation matrix to align an input MRI to a template MRI.
- *Save Common Air:* saves the intermediate output.
- *Reslice:* applies the transformation matrix to register the input MRI.
- *Hard Mean:* calculates the mean of the input MRIs.

Air Reconcile is a unique component that does not produce any data output in this workflow. It merely reconciles the input files. That is why there is only one data input to it with a control input and output. The control parameters ensure that workflow execution does not proceed to Reslice until Air Reconcile has finished executing. Since there is no data flowing through Air Reconcile, this workflow may be considered as a sequential workflow with no parallel branches.

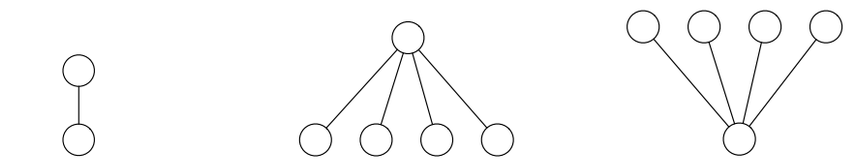
6.4.2 Experimental Setup

The sub-workflows described previously, besides their structural differences, are also differentiated by their *complexity*. We define the complexity of a workflow to be based on two factors; *density* and *richness*. This classification model differentiates the workflow based on their semantics as opposed to their structural or functional characteristics. The density is the number of unknown components in the workflow. A workflow with only one unknown component has *unit density*. The richness of a workflow is defined as the amount of semantic information that is available for the engine to reason with. A workflow with sparse semantic information has low richness and vice versa. These factors together contribute to whether a workflow has *low*, *medium* or *high complexity*. The effect of these parameters on the performance of the engine is explored experimentally. Table 6.1 summarises the relationship between complexity and its factors.

Density	Richness	Complexity
unit	high	low
unit	low	medium
multi	high or low	high

Table 6.1: Relationship between *density*, *richness* and *complexity*.

In general workflows are modelled using specific structural patterns shown in Figure 6.10 [7]. The sub-workflows in the dataset and the structural characteristics they contain are shown in Table 6.2.



(a) Sequential (b) Parallel split (c) Parallel merge

Figure 6.10: Structural workflow patterns.

Sub-Workflow	Characteristics
SW_1	Sequential
SW_2	Sequential
SW_3	Sequential, parallel split, parallel merge
SW_4	Sequential
SW_5	Sequential, parallel split

Table 6.2: Structural characteristics of the chosen dataset.

Based on these patterns, multi-density workflows are further sub-categorised in this research. Workflows in which the unknown components are present in one sequential branch are termed *sequential multi-density workflows*. On the other hand workflows in which the unknown components are present in parallel branches are termed *parallelsaf multi-density workflows*. There are several possible scenarios depending on the number of unknown components and how they are distributed. The following experiments explore the performance of the engine in these scenarios. It will be shown experimentally that the unknown components affect each other in sequential multi-density workflows while they may not interact in parallel multi-density workflows. As discussed in Section 5.2, the parameter used to assess the performance of the component is the MRR.

Since workflows generally comprise the structural patterns shown in Figure 6.10, they can be used to demonstrate the completeness of the engine. All of the structural patterns are contained within the sub-workflows used for this evaluation as shown in Table 6.2. Therefore, it may be concluded that the engine can handle any type of workflow, and is therefore, for the purposes of this research, complete. The next section presents the results of this evaluation and is followed by a discussion of the results.

6.4.3 Results

Exp No	Workflow	Complexity	MRR
1	SW_1	low	0.33
2	SW_1	medium	0.00
3	SW_1	medium	0.25
4	SW_2	low	1.00
5	SW_2	medium	0.00
6	SW_2	medium	0.40
7	SW_4	low	1.00
8	SW_4	medium	0.00
9	SW_5	low	1.00
10	SW_5	medium	0.00

Table 6.3: Sequential Unit-Density Workflow Results

The results shown in Table 6.3 summarise the performance of the algorithm for *sequential sub-workflows* with *unit density*. Each sub-workflow only had one branch of execution and one unknown component that had to be identified. In experiments with low-complexity, there was enough semantic information encoded in the ontology for the system to identify the component. For example, in Exper-

iment 1, *BET* was the unknown component to be identified. Because FLIRT requires a skull-stripped MRI in the *IMG* format, the system was able to suggest BET since it produces a skull-stripped MRI. In comparison, for Experiment 2 the information that *BET* produces a skull-stripped MRI was removed from the ontology. This time the system could not identify BET as a suitable candidate. Therefore it was missing from the suggestions. In Experiment 3, however, the ontology contained the information that BET produces a skull-stripped image. The information that FLIRT takes a skull-stripped image, on the other hand, was removed from the ontology. Therefore, the system assumed that FLIRT can accept any image. Therefore, BET was amongst the suggestions. However, this time there were many irrelevant suggestions, potentially making it difficult for the user to choose the correct component type. The rest of the experiments follow a similar pattern.

Exp No	Workflow	Complexity	MRR
1	SW_1	high	0
2	SW_1	high	0.33
3	SW_2	high	0.14
4	SW_2	high	0
5	SW_4	high	0.5
6	SW_4	high	0
7	SW_4	high	0
7	SW_5	high	0.5
8	SW_5	high	0.5
9	SW_5	high	0.33
10	SW_5	high	0

Table 6.4: Sequential Multi-Density Workflow Results

While Table 6.3 was intended to explore the effects of the level of information present in the ontology on the performance of the algorithm, Table 6.4 explores the effects of having multiple unknown components in the sub-workflow. For example, in Experiment 1, BET and FLIRT were both unknown and had to be identified. The system was able to identify neither correctly, since in the case of FLIRT, there is no component in its downstream that requires a registered image. In the case of BET, 3D B-spline Deformation requires a skull-stripped image. However, since FLIRT is also unknown, the system cannot determine if the semantic requirements can be propagated backwards across it. Therefore, the system cannot identify BET either. In the case of Experiment 2, however, BET and 3D B-spline are the unknown components. BET can be easily identified since FLIRT requires a skull-stripped image. However, 3D B-spine cannot be identified since there is not enough semantic information available. The MRR in this case is the average of the MRR for BET and for 3D B-spline.

Exp No	Workflow	Complexity	MRR
1	SW_3	low	1
2	SW_3	low	0.17
3	SW_3	low	0.14

Table 6.5: Parallel Unit Density Workflow Results

Table 6.5 shows the results of the algorithm for *parallel multi-density workflows*. There are a number of different situations that are possible for this case. The unknown component might be present in one of the parallel branches, or in the sequential part of the workflow. All of these different possibilities are explored in these experiments. For Experiment 1, the Brain Surface Extractor component was unknown and had to be identified. There was enough semantic information from both Partial Volume Classifier and Cerebro (both require a skull-stripped image), to suggest the Brain Surface Extractor component. Experiments 2 and 3 were similar.

Exp No	Workflow	Complexity	MRR
1	SW_3	high	0.16
2	SW_3	high	0.57
3	SW_3	high	0.08
4	SW_3	high	0.04

Table 6.6: Parallel Multi-Density Workflow Results

The last set of experimental results are shown in Table 6.6. These experiments were designed to explore the performance of the algorithm for *parallel multi-density* sub-workflows. For these purposes, it was assumed there was always sufficient information in the ontology to identify the components. The possibilities explored were whether the unknown components existed in parallel branches of execution, or sequential branches of execution. For example, in Experiment 1, Partial Volume Classifier and Cerebro were the unknown components in parallel branches of execution. Since there was sufficient ontological information to determine that they provide segmented images (Cortex and Pial Surface Generator both require the same), the algorithm was able to correctly suggest both components. For Experiment 2, Brain Surface Extractor and Cerebro were unknown. The system was still able to suggest both components since Partial Volume Classifier requires a skull-stripped image. This allowed the system to determine that the Brain Surface Extractor component must be providing that. However, in Experiment 3, in addition to the previous two components, Partial Volume Classifier was also unknown. In this case, the system was able to identify both Cerebro and Partial Volume Classifier, but Brain Surface Extractor could not be identified. It should be noted, however, that once Cerebro and Partial Volume Classifier have been identified, Brain Surface Extractor can be identified in the next iteration. For Experiment 4, all the previous three components as well as Cortex were unknown. For this experiment only Cerebro could be identified. In a subsequent iteration Brain Surface Extractor can also be identified, but it is not possible for the algorithm to identify the other two components.

6.4.4 Discussion

The results shown in the previous section lead us to several conclusions. They show that the algorithm is highly-sensitive to the amount of semantic information available in the ontology. Without sufficient information, it is difficult for the algorithm to provide any meaningful suggestions. Provided sufficient semantic information, the performance of the algorithm also depends on the number and distribution of the unknown components. For example, for sequential unit-density workflows, the algorithm can usually generate meaningful suggestions. The situation becomes more complicated in the case of multi-density workflows. In this case, a component may or may not be identified. Its

success or failure depends upon whether enough known components exist to provide the appropriate semantic context to identify the unknown components. In the following sections, the significance of the results with respect to the structural workflow patterns specified earlier are discussed.

In the case of parallel unit-density workflows, if the unknown component exists in the sequential part of the workflow, then the required semantic information may come from either of the subsequent parallel branches. However, if the unknown component is in one of the parallel branches, then for all intents and purposes it may be thought of as a sequential workflow. Any semantic information in the other parallel branch is irrelevant. However, in parallel multi-density workflows, the situation is more complex. Depending on the distribution of the unknown components, the workflow may be treated as a single sequential workflow, or several sequential workflows. The rules governing the success or failure of the algorithm for each parallel branch remain the same as for sequential multi-density workflows.

6.4.4.1 Sequential Patterns

In a sequential workflow pattern, tasks are executed one after the other with no parallel branches of execution. For unit density sequential workflows, two typical possibilities are shown in Figure 6.11. In Figure 6.11a, the unknown component and the component that provides the necessary semantic information to determine its type (the semantic source) are adjacent to each other. In Figure 6.11b, the two components are not adjacent. Experiment 1 in Table 6.3 is an example of the former scenario while Experiment 4 is an example of the latter scenario. From the results it can be concluded in the case of unit-density workflows, the distance between the unknown component and semantic source is irrelevant. Provided there is enough semantic information, the system is able to suggest a relevant candidate component. This situation is a bit more complicated in the case of multi-density workflows.

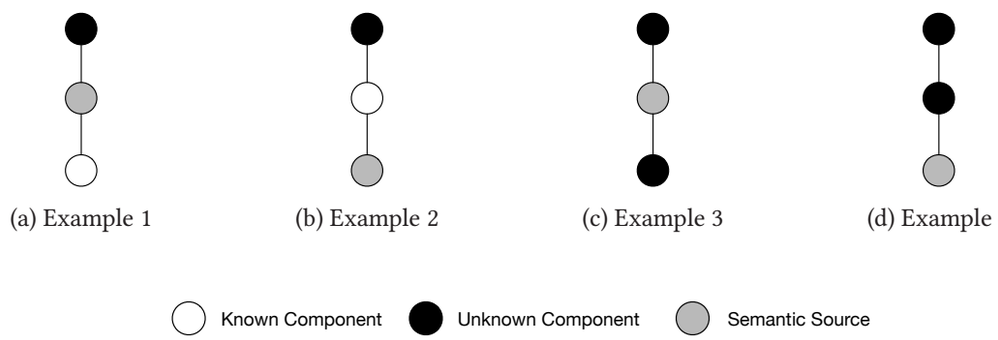


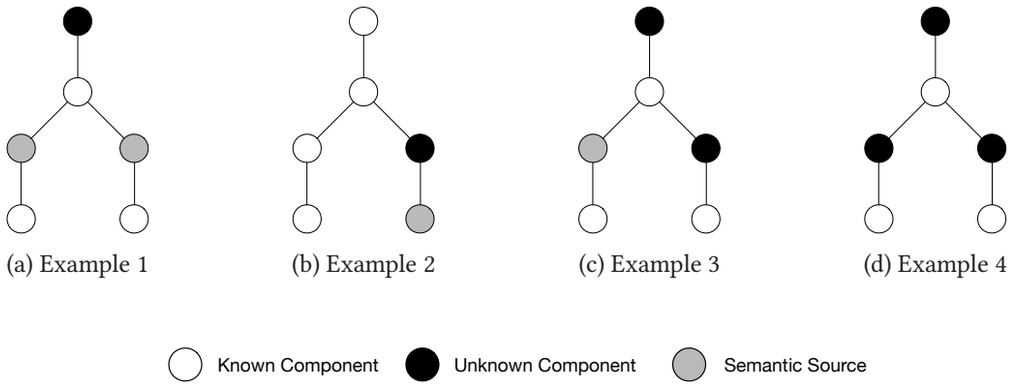
Figure 6.11: Sequential Workflow Examples

For sequential multi-density workflows, Figures 6.11c and 6.11d represent typical scenarios. Experiment 2 in Table 6.4 is an instance of Example 3 and Experiment 1 is an instance of Example 4. As can be seen, the system was unable to identify the unknown component in Experiment 1 but was successful in Experiment 2. This is because even though the semantic source existed in the workflow, the additional unknown component in between the two in the case of Experiment 1 prevented the system from propagating the semantics across. Recall from Chapter 4 that doing so requires a transitive dependency to be specified. Since the middle component is unknown and there is no semantic information about it, the system is unable to determine if semantics can be propagated across it or not.

This shows that in multi-density workflows, the distance between the unknown component and the semantic source may or may not matter.

6.4.4.2 Parallel Split

In a parallel split, a task or sequence of tasks are succeeded by two or more parallel branches of execution. The parallel nature of the workflow affects the performance of the system in various scenarios. Typical scenarios include those shown in Figure 6.12. The scenario shown in Example 1 represents Experiment 1 in Table 6.5. As can be seen the unknown component was easily identified by the system. The system still performed well in Experiment 2, typified by Figure 6.12b. This shows that for the two simple cases where there is only one unknown component, its location in the workflow does not matter as long as there is sufficient semantic information available. As with sequential workflows, the situation is more complex when there are more than one unknown components in the workflow. The multi-density parallel workflow example shown in Figure 6.12c represents the situation



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Figure 6.12: Parallel Workflow Patterns

in Experiment 2 of Table 6.6. The system was still able to identify one component; the one at the top because there was still sufficient semantic information. However, in Experiment 4, represented by Figure 6.12d, the system could not identify any component because sufficient semantic information was not available. This shows that in a parallel split, each branch acts as a separate sequential workflow. As long as sufficient semantic information is available in either branch, components can be identified. The rules regarding the distance between the unknown component and the semantic source remain the same as in sequential workflows.

6.5 End-to-End Evaluation of the Entire Framework

The end-to-end framework evaluation methodology follows the same pattern as the methodology for evaluating the domain ontology update engine, shown in Section 5.5 and also bears similarity with the work of Goderis et al. [8]. They have devised a methodology to study the effectiveness of workflow discovery tools. This methodology is based on behaviour models describing the users' attitudes towards workflow reuse and discovery. As such most of their benchmarks are related to these behaviour models and not relevant to the purposes of this research. One of the benchmarks, however, involves users creating workflows that either contain or are part of other workflows. The

same approach is used to perform end to end evaluation of this framework. The experimental setup used is shown in Figure 6.13. Since the framework works in two phases; the offline and the user-

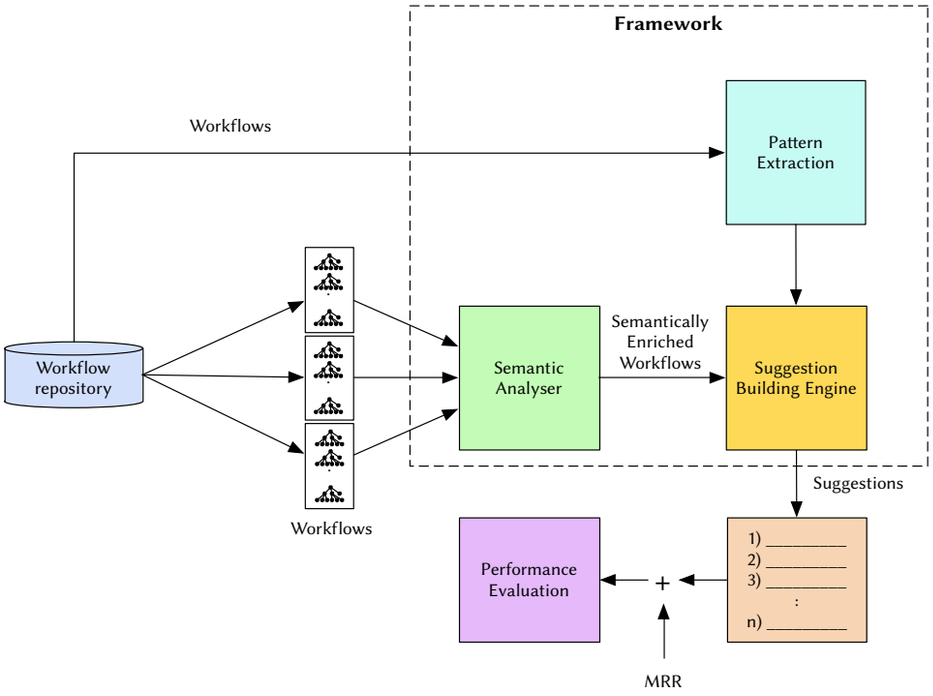


Figure 6.13: End-to-End Evaluation Methodology

initiated processing phases; the evaluation was also carried out in two phases. For the offline phase, workflows from the workflow repository were first input to the pattern extraction component. The extracted patterns were then written into the ontology. For the user-initiated processing phase, three sets of workflows were provided to the framework as input. Within the framework, the workflows were first semantically-enriched. The Suggestion Building Engine then generated the suggestions based on those workflows. Similar to the domain ontology update engine, the MRR coupled with the number of steps required to construct the workflow were used to evaluate the suggestions. A higher MRR indicates that the relevant suggestion was ranked highly by the system. For the number of steps required to construct the workflow, a lower number is desirable. This would indicate that the system was of greater assistance by minimising user involvement. On the contrary, a high number of steps would indicate that the user needed to be more involved in the design process. As with the domain ontology update engine, the completeness of the framework can be demonstrated by evaluating its performance on various types of workflows. Since the structural patterns shown in Figure 6.10 are also contained within the workflows in the repository, it can be said that they represent all types of workflows. Therefore, the framework may be considered complete.

To evaluate the algorithm, 65 neuroscience workflows from the LONI repository were used. During the pattern extraction phase, the minimum frequency threshold for each pattern was set at 4. It was so chosen because it is approximately equal to 5% of the total workflows, which is the significance level commonly used in statistical significance testing [3]. In addition, Closed Graph [9] mining was enabled to eliminate overlapping patterns and reduce the number of overall patterns mined. These patterns were written into the ontology for use by the Suggestion Building Engine. The ontology

ID	Name	Size	In Repository
Wf_1	BrainParser (Hippocampus)	14	✓
Wf_2	BrainParser (56 Structures)	14	✓
Wf_3	Automated ROI Extraction/Volume Calculation	22	✓
Wf_4	BrainParser (Hippocampus)	14	×
Wf_5	BrainParser (56 Structures)	14	×
Wf_6	Automated ROI Extraction/Volume Calculation	22	×
Wf_7	MDT with KLMI (Existing Atlas)	15	⊗
Wf_8	MDT with KLMI (1 Subject Bias)	13	⊗
Wf_9	MDT with KLMI (Multiscale Symmetric)	27	⊗

✓: Workflow is present in repository with overlapping patterns.

×: Workflow is not present in repository but has overlapping patterns with other workflows.

⊗: Workflow is not in repository and has no overlapping patterns.

Table 6.7: Workflows used to perform overall evaluation.

already contained descriptions of the various algorithms along with their inputs and outputs.

The three sets of workflows constituted three categories from the dataset. These were workflows that (a) already existed in the repository, (b) did not exist in the repository but had overlapping patterns with it, and (c) did not exist in the repository and had no overlapping patterns with it. The first two categories are similar to the tasks used by Goderis et al. The third category has been added in order to evaluate the benefits of combining semantics with usage patterns. For the experiments, three workflows of each category were chosen and are shown in Table 6.7. The table also lists their size in terms of the number of components they comprise and whether they were present in the repository or not. For workflows that were not present in the repository, it is also indicated whether there were any overlapping patterns with other workflows. For (b) and (c), the workflows were first removed from the original repository in turn, reducing the total count to 62 workflows. The results of the experiments are presented in the next section.

6.5.1 Results

For each of the workflows provided as input to the framework, Table 6.8 shows the number of steps it took to construct the workflow and the average MRR for each workflow. The average MRR was calculated by averaging the MRRs for each individual step required to construct the workflow using the suggestion framework. Overall it can be seen that the framework requires few steps to construct workflows that have overlapping patterns with other workflows in the repository. For Wf_1 it took only one step to construct since the entire workflow appears as a sub-workflow in other workflows in the repository. Similarly, for Wf_2 it took only two steps. This is possible since the framework does not suggest only one component at a time; instead it can suggest as many as required. For Wf_3 it took 9 steps to complete the workflow even though it consisted of 22 components. This was again made

Workflow	No. of steps	MRR
Wf_1	1	0.14
Wf_2	2	0.20
Wf_3	9	0.22
Wf_4	1	0.17
Wf_5	1	0.18
Wf_6	10	0.20
Wf_7	12	0.50
Wf_8	13	0.18
Wf_9	18	0.17

Table 6.8: Overall results.

possible by the existence of overlapping patterns. For workflows Wf_4, Wf_5 and Wf_6, the number of steps required to complete the workflows were still much less than the sizes of the workflows. This was true even though the workflows themselves did not exist in the repository. The results for workflows Wf_7, Wf_8 and Wf_9 show that even though the workflows did not exist in the repository and did not have any overlapping patterns, the number of steps required to construct them were still less. This is possible because even though there were no overlapping patterns directly, the framework was able to find patterns after generalisation.

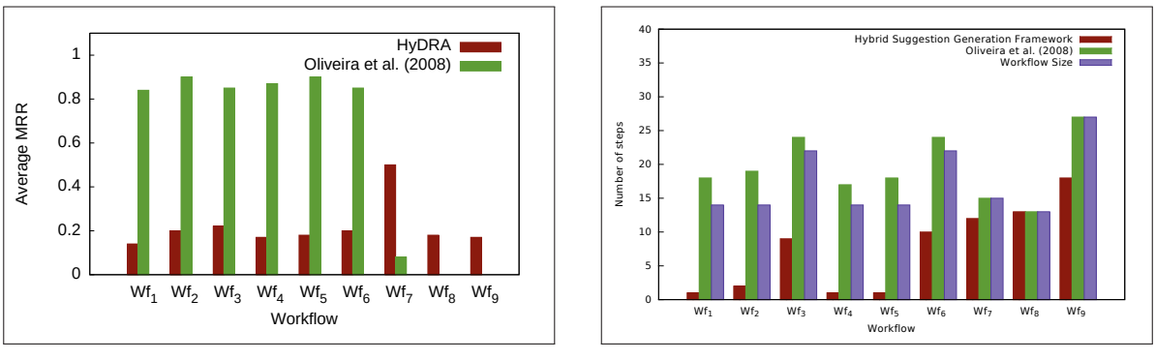
Workflow	No. of steps	MRR
Wf_1	18	0.84
Wf_2	19	0.90
Wf_3	24	0.85
Wf_4	17	0.87
Wf_5	18	0.90
Wf_6	24	0.85
Wf_7	15	0.08
Wf_8	11	0
Wf_9	27	0

Table 6.9: Oliveira results.

Table 6.9 shows the results of experiments run using an alternative system developed by Oliveira et al [10]. It was chosen for comparison because it is a system that does not employ any semantics. It only relies on frequent patterns. Since this research attempts to show how combining semantics with patterns can improve the suggestions, Oliveira et al's system is a suitable candidate for comparison. In contrast to the results shown in Table 6.8, it can be seen that this system works well for workflows that are already present in the repository (Wf_1,Wf_2,Wf_3). The results are similar for workflows that have overlapping patterns with other workflows in the repository. However, the system does not work well for workflows that are not present in the repository and have no overlapping patterns. For Wf_1, Wf_2 and Wf_3, the average MRR was quite high. However, the number of steps required to construct the workflow was equal to or greater than the size of the workflow. This is because the system only

suggests one component at a time. Therefore, even though the accuracy of the system was high, it still required a significant number of steps to complete the workflow.

The reason the number of steps was greater than the size of the workflow in some cases was because some components had multiple links between them. So in some cases it took several steps to connect two components together. For workflows Wf_4, Wf_5 and Wf_6 the results were similar to the results of the previous three workflows because of overlapping patterns. The major difference between HyDRA and this system is highlighted in the results for workflows Wf_7, Wf_8 and Wf_9. In these cases the compared system was not useful in constructing the workflow. This is because there were no patterns in the repository for the system to draw upon for suggestions. And since the system does not consider semantics, it could not make any suggestions.



(a) Average MRR

(b) No. of steps

Figure 6.14: Comparison of HyDRA and Oliveira et al.

Figure 6.14 show a comparison of the two systems with respect to the average MRR and the total number of steps required to construct the workflows. For workflows that are already in the repository or have overlapping patterns, the MRR is low when compared to Oliveira et al. because HyDRA suggests more than one component at a time whereas Oliveira suggests only one. In general, the larger the pattern, the less frequent it is. Since both systems employ frequency-based ranking mechanisms, Oliveira's suggestions occur more frequently, and thus are ranked higher. On the other hand HyDRA's suggestions occur less frequently because they consist of multiple components. Thus they are ranked lower. In the case of workflows with no overlapping patterns, HyDRA clearly outperforms Oliveira. In Figure 6.14b, HyDRA clearly outperforms Oliveira et al.'s system since the fewer steps it takes to construct a workflow, the more efficient the system is.

6.5.2 Discussion

This framework employs semantics in a two-step process to generate suggestions; a) before the pattern mining process to extract generalised patterns, and b) when generating suggestions as an alternative to the pattern-based suggestions. In order to understand the benefits semantics afford in this process, an analysis of the different possible scenarios is necessary. Based on the results shown in the previous section, these scenarios are given below:

1. There are overlapping patterns between the workflow being constructed and the workflows in the repository.
2. There are no overlapping patterns before the workflow has been generalised, but they emerge

post-generalisation.

3. There are no overlapping patterns at all.

This section discusses how the framework performs in each of these scenarios.

6.5.2.1 When Overlapping Patterns Exist

When overlapping patterns exist generating suggestions is quite straightforward for the system. This can be seen by the fact that the number of steps required to construct Wf_1, Wf_2, Wf_3 was very small for this framework compared to their size. For the same workflows, Oliveria's system also performed well. Even though the number of steps required was quite high, the average MRR was also quite high. This shows semantics do not afford any significant benefits in this scenario. However, they do not degrade performance either; HyDRA performs well for this scenario as well. Therefore, it can be concluded that in this incorporating semantics does not provide any improvement over existing systems.

6.5.2.2 When No Overlapping Patterns Exist Before Generalisation

Another possible scenario is when there are no overlapping patterns pre-generalisation. However, patterns may emerge once generalisation is performed. Such an example is shown in Figure 6.15. These examples are taken from the "Atlas-based ROI Analysis for DTI Data" [11] workflow and Wf_7.

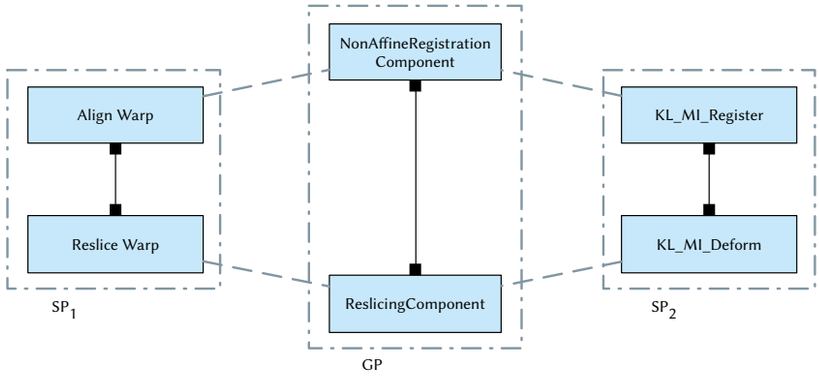


Figure 6.15: Patterns emerging after generalisation.

Both *Align Warp* and *KL_MI_Register* are components of type *NonAffineRegistrationComponent*. *Reslice Warp* and *KL_MI_Deform*, on the other hand, are both components of the type *ReslicingComponent*. Individually the patterns SP_1 and SP_2 do not occur frequently enough to be mined by the pattern miner. After generalisation is performed, however, both these patterns are generalised to the same general pattern GP. Since SP_1 and SP_2 are present in several workflows, the total frequency of GP crosses the minimum frequency threshold and it is picked up by the pattern mining algorithm. Thus when constructing Wf_7 this pattern is suggested. This does not occur in Oliveira as there is no generalisation process involved. Each individual pattern SP_1 and SP_2 remains infrequent. Therefore, the relevant suggestion is ranked relatively lowly by Oliveira during suggestion generation. Ranking a suggestion low indicates that the system has low confidence in its correctness. Moreover, it also decreases the probability that the user will choose that sugggestion, thereby reducing the utility of the system for the user.

6.5.2.3 No Overlapping Patterns

When no overlapping patterns exist either pre or post generalisation, pattern-based systems like Oliveira are unable to make any useful recommendations. However, by leveraging semantics, HyDRA is still able to make recommendations. An example is shown in Figure 6.16. *Reslice* takes a transfor-

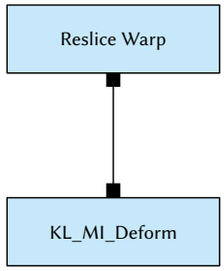


Figure 6.16: An example semantics-based recommendation.

mation matrix from *Align Linear* and applies it to an input image to register it to a template image. The final output is a *RegistrationDataset* in *IMG* format. *KL_MI_Deform* takes a template image in the *IMG* format and registers an input image to it. There are no patterns in the workflow repository that link these two components together. However, by looking at the semantic descriptions of the both components, the system determines that *KL_MI_Deform* can be connected to *Reslice*. Therefore, it suggests the former as a successor to the latter. This is a scenario where Oliveira fails to produce any suggestions. Thus semantics afford a clear advantage here.

These scenarios together clearly demonstrate that semantics, applied in the two-stage fashion this framework adopts, help to generate more accurate recommendations. In one instance patterns emerge after generalisation where there were none before. In another semantics provide additional information where no patterns were available at all. Thus, an improvement is achieved over pattern-based systems. The next section concludes this chapter.

6.6 Summary and Conclusions

This chapter presents an evaluation of the various components in the framework along with an end-to-end evaluation of the overall framework. The various components in the framework include the domain ontology, the pattern extraction component, and the workflow component identification algorithm. In addition to these components is the overall suggestion-building engine. The OntoClean methodology was used to evaluate the domain ontology. This methodology specifies a set of metaproperties that place constraints on the way different concepts in the ontology can be related. Specifically, the constraints place limits on the subsumption relationship. Using this methodology, one can justify various design choices. This is necessary because there is generally no single correct way to design an ontology. For the pattern extraction component, a number of different algorithms were applied on the dataset and the results compared. It was shown experimentally that the results from the algorithms match. Additionally, the results also comply to theoretical patterns that are characteristic of combinational problems such as subgraph enumeration.

In order to evaluate the component identifier, a set of sub-workflows were chosen from the evaluation dataset. These workflows were chosen for their illustrative capabilities to demonstrate various

aspects of this algorithm. Generally, scientific workflows consist of various patterns such as sequential and parallel branches. These experiments were designed to explore the performance of the algorithm for various possibilities. These possibilities arise from the number and distribution of unknown components in the workflow as well as the level of semantic information available in the ontology. The parameter used to evaluate the performance of the algorithm was the MRR. It was shown experimentally that even for parallel branches, the behaviour of the algorithm was equivalent to each branch being treated as a sequential workflow.

The end-to-end evaluation was conducted along the same lines as the evaluation of the component identification algorithm. A set of workflows were chosen from the dataset and constructed using suggestions from the framework. The results were evaluated and compared using two factors; the MRR and the number of steps required to complete the workflow. For comparison, a pattern-based systems developed by Oliveira et al. was also evaluated using the same methodology. Experiments show that there are three scenarios possible when designing workflows. These are a) the workflow being designed has overlapping patterns with other workflows in the repository, b) the workflow being designed does not have any overlapping patterns before generalisation, and c) the workflow does not have any overlapping patterns at all. The results show that HyDRA shows no improved performance in the first scenario. However, it clearly performs better than the compared system in the latter two scenarios. The next chapter concludes this thesis and discusses future work.

Bibliography

[1] N. Guarino and C. A. Welty, "An overview of OntoClean," in *Handbook on ontologies*. Springer, 2009, pp. 201–220.

[2] D. E. Rex, J. Q. Ma, and A. W. Toga, "The LONI pipeline processing environment," *NeuroImage*, vol. 19, no. 3, pp. 1033 – 1048, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105381190300185X

[3] S. L. Chow, *Statistical significance: Rationale, validity and utility*. SAGE Publications Limited, 1997, vol. 1.

[4] Z. Tu *et al.*, "Brain anatomical structure segmentation by hybrid discriminative/generative models," *Medical Imaging, IEEE Transactions on*, vol. 27, no. 4, pp. 495–508, 2008.

[5] D. W. Shattuck and R. M. Leahy, "BrainSuite: an automated cortical surface identification tool," *Medical image analysis*, vol. 6, no. 2, pp. 129–142, 2002.

[6] "fMRI preprocessing using Air," http://users.loni.usc.edu/~pipeline/serverlib/view_workflow.php?file=cranium/fMRI/Groups/fMR_using_AIR.pipe [Last accessed 30th Dec, 2014].

[7] L. Ramakrishnan and B. Plale, "A multi-dimensional classification model for scientific workflow characteristics," in *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, ser. Wands '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:12.

[8] A. Goderis *et al.*, "Benchmarking workflow discovery: a case study from bioinformatics," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 16, pp. 2052–2069, 2009. [Online]. Available: http://dx.doi.org/10.1002/cpe.1447

[9] X. Yan and J. Han, "CloseGraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 286–295.

[10] F. T. Oliveira *et al.*, "Provenance and annotation of data and processes," ser. Lecture Notes in Computer Science, J. Freire, D. Koop, and L. Moreau, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Using Provenance to Improve Workflow Design, pp. 136–143.

[11] "Atlas-based ROI analysis for DTI data," http://bit.ly/1bIrtxK [Last checked: 7th July, 2014].

CHAPTER 7

Conclusions and Future Work

7.1 Introduction

This thesis explores the extent to which semantics can be combined with frequent usage patterns by workflow recommender systems to generate suggestions. In this regard a framework has been developed that can a) infer the semantics of unknown components in a workflow, and b) assist scientific users when designing workflows by suggesting workflow components. HyDRA comprises a semantic inference as well as a semantic analysis algorithm that support the suggestion generation process (cp. Chapter 4). The framework is intended to be used by scientific users that wish to run workflow-based data analyses on distributed infrastructures. For example, referring to the neuGRID example introduced in Section 1.2.1, a neuroscientist studying neurodegenerative diseases may wish to compare the MRIs of several patients to determine if a particular patient shows signs of suffering from such a disease. Such comparisons are typically compute-intensive and thus ideally suited for workflow-based execution on distributed infrastructures. Moreover, this research also helps in addressing the problem of transference of expertise also introduced in Section 1.2.1. Less technically-knowledgeable users can take advantage of the expertise of advanced suers via the mechanisms employed by this body of work.

Workflow design poses some challenges for users including selecting appropriate components and specifying links correctly between them. HyDRA can intelligently assist such scientists design their workflow-based analyses. This scenario is typified in Figure 7.1. HyDRA, once integrated in a workflow composition tool, can communicate with a component repository as well as a semantic repository to generate suggestions for the users as they are constructing the workflows. The framework helps alleviate the aforementioned challenges. Moreover, it also assists the users in keeping the semantic repository up-to-date by identifying unknown components and inferring their semantics.

This chapter concludes the thesis by answering the various research questions formulated in Chapter 1, culminating in the resolution of the stated research hypothesis.

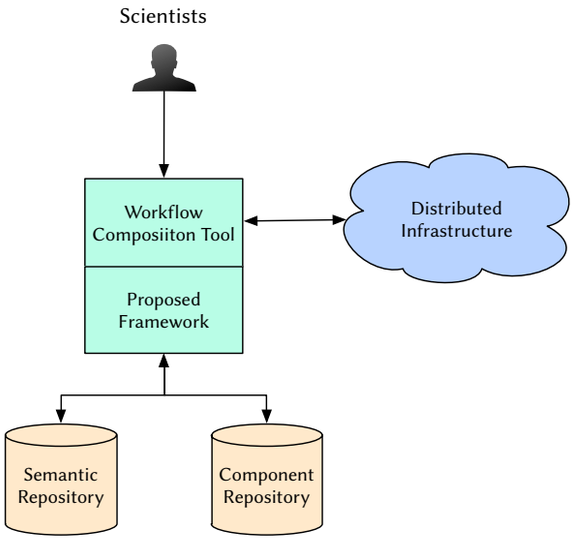


Figure 7.1: Proposed framework in practice.

7.2 Answering the Research Questions

The following research questions were posed in Chapter 1:

Question 1. *What are the limitations of current workflow composition and related systems?*

This thesis describes the workflow composition process with the intention of identifying steps where intelligence can be applied to assist users when designing workflows (cp. Section 2.2). These include constructing correct workflows, finding appropriate components for their requirements, selecting parameter values, determining incompatibilities between components and identifying usage patterns in workflows for future suggestions. Most workflow management systems only address some of these challenges (cp. Section 2.3). They attempt to ensure workflow correctness by ensuring that input and output data types of connected components match such as LONI, Triana and Taverna [1, 2, 3]. They generally do not take into account any additional information. In addition, most systems only provide simple text-based searching of components in the repository. However, some systems exist that add intelligent suggestion features to existing workflow composition systems. Such systems are termed recommender systems in literature.

This research classifies recommender systems into three categories according to the approach they adopt to generate suggestions for users and discusses the pros and cons of each category (cp. Section 2.5). The three categories are ratings-based systems, pattern-based systems and semantics-based systems. Rating-based systems employ explicit ratings for items provided by users and attempt to find other items that the user may be interested in. Pattern-based systems attempt to identify patterns in the way users use a system. They then suggest items to users based on these patterns. Semantics-based systems attempt to suggest semantically-compatible items. Ratings-based systems are not suitable for workflow composition since they attempt to find items based on similarity which does not apply to workflows. Regarding pattern-based systems it was concluded that they work for popular or frequently used items; but they omit rare items that may actually be more useful for the user. Semantics-based systems, on the other hand, are good at finding all items that the user may need, but cannot determine

which of them are more appropriate in a particular scenario. Moreover, they require significant effort to set up initially.

A system that addresses the aforementioned drawbacks, therefore, should have the following properties:

1) It should include additional information (such as semantics) when determining inter-component compatibility.
2) It should not neglect rare items if they are more suitable for users.
3) The suggestions generated should be dynamic and adapt with the availability of more information.
4) It should not require significant effort to set up.

A framework that addresses these points has been designed in this research by combining semantics and pattern-based approaches. Patterns can help find useful items for users based on historical data. Semantics can fill in the gap when relevant historical data is not available. The comparision of HyDRA with Oliveira et al.'s system confirms the usefulness of incorporating semantics (cp. Figure 6.14). Moreover, they can also help inform the pattern extraction process to improve it.

Question 2. *To what extent can workflow component semantics be used to improve the suggestions?*

Existing systems use semantics only to determine compatibility between components (cp. Section 2.5.3). In HyDRA, semantics are applied in two phases to generate suggestions when constructing workflows. In the first phase the set of workflows in the repository are generalised so that functional units of workflow components can be mined. In the second phase, the semantics are used to suggest compatible components during workflow creation. Another way in which semantics are utilised in this research is to infer the semantics of unknown components in a workflow as shown in Section 6.4.3. Doing so enables the semantic repository to be automatically updated without user intervention. All three use cases of semantics are discussed in Chapter 4. These use cases together help us to answer Research Question 2 and show how semantics can be utilised by workflow recommender systems. Moreover, Figure 6.14 shows a comparison of the performance of HyDRA with a similar system that does not employ semantics. Results show a clear improvement in two out of three scenarios as discussed in Section 6.5.2;

1) When there exist overlapping patterns between the workflow under construction and the workflow repository. In this scenario semantics do not improve the quality of suggestions.
2) When there are no overlapping patterns before generalisation. In this case generalisation may result in overlapping patterns emerging.
3) When there are no overlapping patterns after generalisation. In this case the system may still able to offer suggestions based on semantic compatibility.

Question 3. *What approaches are suitable for mining historical usage patterns and how can they be used to improve suggestions?*

Pattern-based systems employ two main approaches to mining patterns. They either extract subgraphs from workflows or extract individual links based on frequency of occurrence. Extracting subgraphs allows the system to generate suggestions with little computational overhead but the pattern-mining process is expensive. On the other hand mining individual links is computationally economical but the suggestion generation process is complicated if more than one component is to be suggested.

Because the user experience is more concerned with the efficiency of the suggestion generation process, this research has focused on mining entire subgraphs. Several subgraph mining algorithms were evaluated and results show that gSpan with Closed graph mining is both efficient and mines complete patterns. This helps us to answer the first part of the question and is discussed in more detail in Section 2.9. Semantics by themselves can only enable reasoning based on static information unless upgraded manually by users. Including frequent usage patterns in the process allows the suggestions to be dynamic and history-based. The performance of HyDRA has been evaluated when historical usage patterns are present in comparison with its performance when they are not (cp. Table 6.8). The results empirically prove the benefit of using historical usage patterns as they both improve the MRR of the suggestions as well as reduce the number of steps required to construct a workflow. This dynamism improves the suggestion generation process and helps us to answer the second part of the question.

Question 4. *How can workflow component semantics and historical usage patterns be combined to improve the suggestions?*

Existing approaches adopted by workflow recommender systems employ either semantics or frequent usage patterns to generate suggestions. In this research both of these techniques have been combined. The semantics are used to mine generalised frequent workflow patterns from the repository. Generalising the patterns allows the system to mine patterns that otherwise may not have been picked up. The reason for this is expounded upon in Section 3.3.8. This section illustrates how semantics can be combined with patterns to generate suggestions. The quality of the suggestions themselves is evaluated empirically in Section 6.5. It has been shown in Figure 6.14 that combining semantics with frequent usage patterns improves the quality of the suggestions generated by the system. Improving the quality of suggestions in turn improves the utility of the recommender system for the user. It helps users to construct and design complex workflows in a time-efficient and reliable manner. Moreover, it also reduces the effort required by users to construct workflows.

7.3 Answering the Research Hypothesis

The research hypothesis is restated here for convenience:

"Workflow component semantics along with their historical usage patterns can be used to improve the suggestions offered by recommender systems."

Answering Research Questions 2 and 3 allows us to conclude that semantics and workflow patterns can be used separately to generate suggestions. By answering Research Question 4, it can be concluded that together, semantics and workflow patterns can be used to improve the quality of the suggestions generated. This fact has also been proven empirically in this research. Therefore, the hypothesis can be said to have been proven true. Thus, a new category of recommender systems can be said to have been created. This thesis had earlier classified recommender systems into three categories; rating-based, pattern-based and semantics-based. The new category comprises HyDRA that combines the properties of pattern-based and semantics-based systems. It has also been shown in Chapter 6 that HyDRA offers an improvement over existing systems.

HyDRA can assist users in several scenarios where traditional workflow design recommender systems cannot:

1) When the relevant component is rare.
2) When functional groups of components are more frequent than specific groups of components.
3) When the semantics of some components in a workflow are unknown.

In the case of 1) a component that is rare may not be picked up by the pattern extraction process since it does not satisfy the minimum frequency of occurrence criterion. In 2) groups of components often perform a composite function and thus co-occur frequently. However, they may not be frequent enough to satisfy the frequency threshold. Often there are several components that perform the same function but are appropriate in different circumstances. Such alternative components may perform the same composite function as another group of components. Using semantics, HyDRA can identify such alternative components and treat them as the same group, consequently rendering it sufficiently frequent to be extracted as a pattern. This would not be possible in traditional systems. In 3) the system uses the semantics of surrounding components to infer the unknown semantics. This can help keep the semantic repository up-to-date. The next section presents the key contributions of this research.

7.4 Key Contributions

This section lists and summarises the key contributions of this research :

A Hybrid Suggestion Generation Approach: this research shows how semantics can be used along with frequent usage pattern extraction to improve the quality of suggestions. Semantics-based systems cannot function until semantics about the components in the repository have been specified. Specifying the components semantically is a time-consuming task and requires concentrated efforts by users. In addition, semantics-based systems are static and do not learn as they are used. Therefore, unless the users put in the time and effort to specify the semantics of the components, the system will not be able to generate suggestions. Semantics-based systems are very good at determining which components are compatible with each other. However, since they do not incorporate frequent usage patterns, in situations where there are a number of similar components to choose from, they cannot prioritise the suggestions. Therefore, important expert knowledge encapsulated in the frequent usage patterns of other users is missing. In addition, two components may be semantically compatible; however the way they are implemented may make them unsuitable for use together. Semantics-based systems would be unable to determine this incompatibiity unless it is explicitly specified. Pattern-based systems cannot generate suggestions unless sufficient data is available for them to find useful patterns.

In addition, since pattern-based systems prune suggestions and only keep the most frequent ones, rare cases where the less frequently used components are more appropriate for the user's needs get ignored. Therefore, in this scenario, the suggestions would be both less relevant as well as misleading for the user. Combining both approaches helps to address the drawbacks of each.

Using Semantics to Improve the Quality of Suggestions: this research applies semantics in two phases. In the first phase, the usage patterns are combined with a functional taxonomy of workflow components to identify generalised patterns in the workflows. These patterns allow the system to attempt and infer what a user is attempting to do to provide contextual recommendations.

Shortcomings in existing generalisation approaches are also identified and addressed in this research with specific regards to workflows. In the second phase, the semantics of the workflow components are utilised to identify compatibility between components. A novel semantic analysis mechanism is introduced that ensures that the system takes into account sufficient semantic information to produce context-sensitive suggestions. The lack of this semantic analysis mechanism in existing approaches means that the suggestions are generated in the absence of a useful context.

Updating the Ontology: this research also addresses the question of how the ontology can be kept up to date by introducing a novel semi-automated methodology to detect and identify new components. A semantic inference algorithm has been devised in this regard that infers the semantic properties of unknown components in a workflow. It attempts to do so by analysing the known semantic properties of other components in the workflow, and then by inferring the most likely properties of the unknown component. The ontology is then queried on the basis of those inferred properties and all possible candidates are returned to the user.

Inferring the Semantics of Unknown Components: for the semantic inference algorithm, provided sufficient semantic information is available, the performance of the algorithm also depends on the number and distribution of the unknown components. For example, for workflows with only one branch and one unknown component, the algorithm can usually generate meaningful suggestions. The situation becomes more complicated in the case of workflows with multiple unknown components. In this case, a component may or may not be identified. Its success or failure depends upon whether enough known components exist to provide the appropriate semantic context to identify the unknown components.

In the case of workflows with parallel branches and one unknown component, if the unknown component exists in the sequential part of the workflow, then the required semantic information may come from either of the subsequent parallel branches. However, if the unknown component is in one of the parallel branches, then for all intents and purposes it may be thought of as a sequential workflow. Any semantic information in the other parallel branch is irrelevant. However, in parallel workflows with multiple unknown components, the situation is more complex. Depending on the distribution of the unknown components, the workflow may be treated as a single sequential workflow, or several sequential workflows. The rules governing the success or failure of the algorithm for each parallel branch remain the same as for sequential workflows.

Comparative Evaluation: comparison against a pattern-based system shows that semantics combined with usage patters can indeed be used to improve the quality of suggestions. The comparison was performed for three scenarios;

a) When there exist overlapping patterns between the workflow under construction and the workflow repository. In this case generating suggestions is quite straightforward for the both systems. There was not much difference in the results for both systems in this scenario. Thus shows semantics do not afford any significant benefits in this scenario. However, they do not degrade performance either; HyDRA performs well for this scenario as well.

b) When there are no overlapping patterns before generalisation. The process of generalisation replaces workflow components with their abstract components that represent the functions those components perform. This process may result in new patterns emerging that did not exist before since there may be many different types of components that perform the same functions. In this scenario, HyDRA would be able to extract those patterns, while the compared system could not. Thus, incorporating semantics in this case improves the quality of the suggestions.

c) When there are no overlapping patterns after generalisation. In this case the system may still able to offer suggestions based on semantic compatibility. The compared system, since it does not take into account semantics at all, would not be able to offer any suggestions int his case.

Having discussed the key contributions of this research, a critical analysis of the same is presented in the next section.

7.5 Critical Analysis

While the research presented herein addresses some important questions, there is some room for improvement in the way they are addressed. These are identified subsequently:

1) Currently the process of inferring the semantic properties of an unknown component considers only one workflow. This may result in properties that the component does not actually posses or properties that are not helpful because they are too generic. To mitigate this effect generic properties that do not provide any meaningful information regarding the identity of the component are filtered. However, some extraneous properties may still be left over. A more robust method of filtering extraneous properties may be to consider several workflows containing the component instead of just one. In this approach, the semantic inference algorithm could be run on all workflows containing the unknown component. Selecting only the most common subset of properties might result in a more accurate set of properties of the unknown component. Formally, if P_i is the set of all semantic properties inferred for the unknown component in workflow i, then $\bigcap_{i=1}^{n} P_i$ would give the most common subset of properties.
2) In order for the semantic analysis algorithm to propagate semantic properties across workflow components, a relationship between the inputs and outputs needs to be specified (cp. Section 4.5.3). This allows the algorithm to avoid ambiguities when a component has multiple inputs or outputs. Currently this relationship is specified manually by domain experts. It may be possible, however, to infer this relationship automatically by analysing multiple workflows. If a particular property always appears at a particular input and output of a component in different instances, then it can be inferred that that property is being transferred from the input to the output. The algorithm can then encode that relationship into an ontology. This would remove some of the burden from domain experts to specify the semantics of workflow components.
3) The semantic analysis process propagates workflow semantics across components. This provides the system with additional information when generating suggestions and allows them to be more accurate. Currently only a single workflow is considered for this process. Considering multiple

workflows may improve this process as well. The algorithm could select the most common subset of semantic properties available after semantic analysis. This could filter extraneous information that the algorithm considers when generating suggestions.

4) When suggestions are being generated, the system converts the partial workflow to a graph. It then searches for patterns in the repository that contain the converted subgraph. However, it may be possible that patterns that contain only subsets of the partially-constructed workflow exist. By searching the repository for patterns that have overlapping subgraphs with the partially-constructed workflow, the quality of suggestions might further be improved.

In the next section future avenues for research are identified.

7.6 Future Directions

The following potential future directions have been identified during the course of this research:

7.6.1 Rank Suggestions Based on Semantics and Frequency of Occurrence

Suggestions are currently ranked first by frequency of occurrence. Suggestions generated by semantic compatibility are not ranked at all. One of the future directions for this research may be to develop a composite method of ranking suggestions based on both frequency of occurrence and semantic compatibility. For example, to retrieve semantically-compatible components, the system first analyses a partial workflow to propagate semantic properties across components. The propagated properties are then used to retrieve semantically-compatible components from the ontology. In order to improve accuracy, some propagated properties are pruned (cp. Section 4.5.5.3). However, there may still be semantic properties left that do not contribute to the components retrieved. Therefore, a composite mechanism can be developed that ranks the semantically-generated suggestions according to the number of semantic properties they satisfy as well as the frequency of occurrence. Such a composite approach may help to rank relevant suggestions higher than frequency-based methods, thereby improving the quality of the suggestions.

7.6.2 Assign Weights to Extracted Patterns Based on Expertise of Authors

It was argued in Section 2.6.3 that different authors with different expertise levels should be given proportional weights when extracting patterns from their workflows. The reason for this is that novice users are likely to make more mistakes when creating workflows. By indiscriminately including all workflows in the pattern extraction process, the potential for adding noise is introduced. To alleviate this shortcoming, it may be desirable to assign a score to every pattern that is extracted. The score can be a weighted sum of the various workflows the pattern was extracted from based on author expertise. This score can be used when generating suggestions to rank them, thus ranking workflows from unreliable authors lower than those from reliable authors. This can help in reducing the noise in the extracted patterns and making the suggestions more reliable.

7.6.3 Parameter Suggestions

Parameter values are an important aspect when constructing workflows. The correct functioning of workflows often depends on appropriate parameter values. In this research only file parameters are considered. However, they may be of many different types such as string, integer, floating point numbers etc. Often the correct values of certain parameters depend on the values of other parameters. These dependencies between parameters can be captured in the ontology by domain experts and can then be used by a parameter value extraction algorithm to extract conditional frequently-used values for those parameters. Furthermore, some distance metric such as the Hamming distance [4] between the values of the dependent parameters in the repository and the partial workflow being constructed can be used. This distance measure can be used to derive a confidence measure for the suggested values that can then be used as a ranking mechanism as well. Doing so would ensure that context-aware values for parameters are extracted. Therefore, context-appropriate suggestions can then be generated for parameter values.

7.6.4 Workflow Correctness

It is important for a workflow system to be able to tell users that the workflows they are designing are correct. There are two aspects to correctness; structural correctness and semantic correctness. Structural correctness ensures that all components are connected properly and that there are no cycles in the workflow. Semantic correctness ensures that all the components connected to each other are semantically compatible; the data types of their inputs and outputs match. Semantic correctness has been addressed in this research. However, structural correctness is also important. Debugging a large workflow can be a tedious and time-consuming task. A system can be designed that takes this aspect into account when generating suggestions using the proposed mechanisms. Such a system can ensure that the suggestions it generates lead to correct workflows. While constructing a workflow, such a system could, for example, check that no cycles have been introduced. Another check could be that all the parameters are connected to appropriate inputs. This approach would make the tedious task of debugging large workflows simpler by breaking it down into discrete, incremental chunks, thereby improving the usability of the system.

7.6.5 Goal State

In addition to specifying how different parts of a workflow fit together, semantics can also be used to specify the initial or final conditions of a workflow. Some work has already been done in this regard [5]. In HyDRA, a description of the final output that is expected of the workflow could help guide the suggestion generation process at each step. For example, given a partial workflow, the system could backtrack from the description of the final output to determine what components would be required to achive the desired output. The various options could be included in the suggestions.

7.6.6 Relevance of Results

For any suggestion system, the ultimate goal is to assist the user in achieving their final result. For a spelling suggestion system, the user must get the word they are looking for. For a search engine, it is important that users get all the relevant results that they require. Similarly, for a workflow composition

system, it is important that the final workflow yield results that the users require or might find useful. Therefore, if a particular workflow provides the required results, giving it a higher weight would bias the results towards successful workflows. Without this, all workflows would have equal weighting. However, not all workflows provide results with the same level of relevance. Therefore, biasing the suggestions towards more relevant workflows would improve them. Such a system could incorporate the relevance of results into the suggestion ranking algorithm.

In conclusion, HyDRA combines semantics and frequent usage patterns for workflow design recommendations. The framework on the one hand helps keep the semantic repository up-to-date. On the other it uses semantics to inform the pattern extraction process. This allows the system to mine patterns in situations where traditional pattern-based systems cannot. Semantics are also used to suggest components in situations where the relevant component is rare, which is not possible for traditional pattern-based systems. However, this framework has primarily been designed for grid/cluster-based distributed systems. A potential future direction could also be to investigate the feasibility of this research for cloud-based distributed infrastructures and the challenges they afford. Moreover, another possible avenue could be the automatic or semi-automatic generation of the ontology supporting this framework.

Bibliography

[1] T. Oinn *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/20/17/3045.abstract

[2] D. E. Rex, J. Q. Ma, and A. W. Toga, "The LONI pipeline processing environment," *NeuroImage*, vol. 19, no. 3, pp. 1033 – 1048, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105381190300185X

[3] I. Taylor *et al.*, "The Triana workflow environment: Architecture and applications," in *Workflows for e-Science*, I. J. Taylor *et al.*, Eds. Springer London, 2007, pp. 320–339, 10.1007/978-1-84628-757-2_20. [Online]. Available: http://dx.doi.org/10.1007/978-1-84628-757-2_20

[4] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[5] J. Kim, A. Gil, and M. Spraragen, "A knowledge-based approach to interactive workflow composition," in *In Proceedings of the 2004 Workshop on Planning and Scheduling for Web and Grid Services, at the 14th International Conference on Automatic Planning and Scheduling (ICAPS 04)*, 2004.

APPENDIX A

Applying the OntoClean Methodology

DatasetParameter (+I, ~R, +U)

A *DatasetParameter* is an MRI that is identified by the patient it belongs to and the method used to obtain it. Therefore, it carries unique identity criteria (+I). Moreover, all types of MRIs have the same identity criteria. A *DatasetParameter* may become a different type of dataset as it is processed a by workflow. For example, it may be an unsegmented dataset at one point and become a segmented dataset at another. Therefore, it is anti-rigid (~R). An MRI is an indivisible whole with no distinct parts, so it has unity (+U). Different types of dataset parameters are shown Table A.1.

Component Type	Metapropertiers	Description
Registration DatasetParameter	+I, ~R, +U	A *RegistrationDatasetParameter* is an MRI that is produced by a *RegistrationProcessing Component*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).
SkullStripped DatasetParameter	+I, ~R, +U	A *SkullStrippedDatasetParameter* is an MRI that is produced by a *SkullStripping Component*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).
NonSegmented DatasetParameter	+I, ~R, +U	A *SkullStrippedDatasetParameter* is an MRI that has not been segmented and serves as input to a *SegmentationComponent*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).

Component Type	Metapropertiers	Description
Reference DatasetParameter	+I, ~R, +U	A *ReferenceDatasetParameter* is an MRI that is used as a reference point for registering another MRI and serves as input to a *RegistrationProcessingComponent*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).
Segmented DatasetParameter	+I, ~R, +U	A *SegmentedDatasetParameter* is an MRI that has been segmented and serves as the output to a *SegmentationComponent*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).
Floating DatasetParameter	+I, ~R, +U	A *FloatingDatasetParameter* is an MRI that is being registered to a reference image and serves as input to a *RegistrationProcessingComponent*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).
Reconstructed DatasetParameter	+I, ~R, +U	A *ReconstructedDatasetParameter* is an MRI that has been reconstructed and serves as the output to a *ReconstructionComponent*. Since it is an MRI, it carries identity criteria (+I), is anti-rigid (~R) and had unity (+U).

Table A.1: *DatasetParameter* components and their OntoClean metapropertiers.

DatasetProcessingComponent (+I, +R, -U)

A *DatasetProcessingComponent* consists of an executable algorithm along with some inputs and outputs. Since each component has certain characteristics that clearly distinguish one from the other such as name, function and version of the executable algorithm, it carries identity criteria (+I). A particular dataset processing component always remains the same component, hence it is rigid (+R). Since a dataset processing component consists of parts that are wholes but different, it carries no unity (-U). In general, all types of dataset processing components posses the same metaproperties. They are shown in Table A.2.

Component Type	Metapropertiers	Description
Statistical ProcessingComponent	+I, +R, -U	A component that performs some statistical processing on an MRI and outputs the statistics.
Registration ProcessingComponent	+I, +R, -U	A component that registers one MRI to another.
Arithmetical ProcessingComponent	+I, +R, -U	A component that performs some arithmetical processing on between two MRIs.
SkullStrippingComponent	+I, +R, -U	A component that extracts the brain and other tissue from the skull in an MRI.
SegmentationComponent	+I, +R, -U	A component that segments an MRI and identifies various brain regions.
ReconstructionComponent	+I, +R, -U	A component that reconstructs MRIs and outputs reconstructed images.
QuantitativeParameter EstimationComponent	+I, +R, -U	A component that estimates a specific physiological parameter such as regional cerebral blood volume and absolute proton density etc.
ResamplingComponent	+I, +R, -U	A component that resamples an input MRI and produces a resampled MRI. Examples of resampling include changing or orientation of an image.
RestorationComponent	+I, +R, -U	A component that restores an input MRI be reducing defects that degrade an image.

Table A.2: *DatasetProcessingParameter* components and their OntoClean metapropertiers.

APPENDIX B

Sample Workflow Analysis XML

```
<module id="BiasFieldCorrector_1"  exists="true">
    <input  id="inputfile"  type="IMG" transitive-to="outputfile">
    <properties>
        <property>
            DatasetParameter
        </property>
    </properties>
    </input>
    <output  id="outputfile"  type="IMG" transitive-from="inputfile">
        <properties>
            <property>
                DeNoisedDatasetParameter
            </property>
        </properties>
    </output>
</module>
<module id="BET1_1" exists="false">
    <input  id="inputfile"  />
    <output  id="outputfile"></output>
</module>
<module id="FLIRT_1" exists="true">
    <input  id="  initialmatrixfile  "  type="XFM">
    <properties>
        <property>
            DatasetParameter
        </property>
    </properties>
```

```
    </input>
    <input id="input" type="IMG" transitive-to="output">
    <properties>
        <property>
            DatasetParameter
        </property>
        <property>
            SkullStrippedDatasetParameter
        </property>
    </properties>
    </input>
    <output id="output" type="IMG" transitive-from="input">
        <properties>
            <property>
                RegistrationDatasetParameter
            </property>
        </properties>
    </output>
    <input id="reference" type="IMG">
    <properties>
        <property>
            ReferenceDatasetParameter
        </property>
    </properties>
    </input>
</module>
```