

SEMANTICS-ENRICHED WORKFLOW  
CREATION AND MANAGEMENT SYSTEM  
WITH AN  
APPLICATION TO DOCUMENT IMAGE  
ANALYSIS AND RECOGNITION

CHRISTIAN CL AUSNER

School of Computing, Science & Engineering

University of Salford, Salford, UK

Submitted in Partial Fulfilment of the Requirements of the  
Degree of Doctor of Philosophy, May 2019

# Abstract

Scientific workflow systems are an established means to model and execute experiments or processing pipelines. Nevertheless, designing workflows can be a daunting task for users due to the complexities of the systems and the sheer number of available processing nodes, each having different compatibility/applicability characteristics.

This Thesis explores how concepts of the Semantic Web can be used to augment workflow systems in order to assist researchers as well as non-expert users in creating valid and effective workflows.

A prototype workflow creation/management system has been developed, including components for ontology modelling, workflow composition, and workflow repositories. Semantics are incorporated as a lightweight layer, permeating all aspects of the system and workflows, including retrieval, composition, and validation.

Document image analysis and recognition is used as a representative application domain to evaluate the validity of the system. A new semantic model is proposed, covering a wide range of aspects of the target domain and adjacent fields. Real-world use cases demonstrate the assistive features and the automated workflow creation. On that basis, the prototype workflow creation/management system is compared to other state-of-the-art workflow systems and it is shown how those could benefit from the semantic model.

The Thesis concludes with a discussion on how a complete infrastructure based on semantics-enriched datasets, workflow systems, and sharing platforms could represent the next step in automation within document image analysis and other domains.

## Table of Contents

Abstract .....	ii
Acknowledgements.....	ix
List of Terms and Abbreviations .....	x
1 Introduction .....	1
1.1 Processes and Automation .....	2
1.2 Relation to Document Image Analysis.....	3
1.3 Motivation .....	5
1.4 Objectives and Methodology .....	6
1.5 Main Contributions .....	8
1.6 Publications .....	8
1.7 Thesis Overview.....	11
1.8 Summary .....	12
2 Automation and Scientific Workflows .....	14
2.1 Workflows and Distributed Computing.....	14
2.2 Workflow Systems .....	15
2.2.1 Taverna .....	19
2.2.2 Kepler.....	21
2.2.3 Triana .....	23
2.2.4 Pegasus.....	24
2.2.5 ASKALON .....	24
2.3 Workflow Languages and Modelling Paradigms.....	26
2.4 Workflow Repositories .....	27
2.5 Automation and Semantic Features.....	28

2.5.1	Semantics in the WINGS Workflow System.....	31
2.5.2	Semantic Workflow Composition Based on Petri Nets.....	32
2.5.3	Semantic Features of the ASKALON Workflow System.....	34
2.5.4	Semantic Grid .....	36
2.6	Planning Algorithms .....	38
2.7	System Design and Advanced Features .....	39
2.8	Workflows in Document Image Analysis .....	40
2.9	Summary .....	42
3	A Semantic Model for Document Image Analysis and Recognition .....	43
3.1	Knowledge Representation and Semantics .....	43
3.2	Ontology Engineering .....	44
3.2.1	Ontology Creation Methodologies.....	46
3.2.2	Meta-Ontology, Formal Language and Expressivity.....	48
3.2.3	Evaluation of Ontologies .....	50
3.2.4	Ontology Maintenance.....	51
3.3	Document Image Analysis and Recognition.....	51
3.3.1	Overview.....	52
3.3.2	Typical Processing Steps .....	54
3.3.3	Data Formats.....	55
3.3.4	Applications .....	55
3.4	Ontology for Document Image Analysis .....	58
3.4.1	Specification .....	58
3.4.2	Knowledge Acquisition .....	58
3.4.3	Conceptualisation.....	59

3.4.4	Integration .....	63
3.4.5	Implementation .....	64
3.4.6	Evaluation .....	65
3.5	Summary .....	65
4	Designing a Semantics-Enriched Workflow System .....	67
4.1	Workflow Model .....	67
4.1.1	Concept of Activity .....	68
4.1.2	Control Flow .....	69
4.1.3	Data and Dataflow .....	72
4.1.4	Workflow Templates .....	75
4.1.5	Semantic Layer .....	76
4.1.6	User Groups and Perspectives .....	77
4.2	Repositories .....	79
4.3	Workflow Creation .....	79
4.3.1	Activity Matching .....	80
4.3.2	Workflow Validation .....	81
4.3.3	Automation .....	81
4.4	Ontology Management .....	83
4.5	Summary .....	84
5	System Implementation .....	85
5.1	System Overview .....	87
5.2	Ontology Editor .....	90
5.2.1	User Interface .....	90
5.2.2	Ontology Data Format .....	91

5.2.3	Ontology Migration Rules .....	92
5.3	Repository Hub .....	94
5.3.1	Workflow Repositories .....	94
5.3.2	Data Repositories .....	98
5.4	Workflow Editor .....	99
5.4.1	Data Ports .....	100
5.4.2	Activities .....	103
5.4.3	Semantic Labels .....	108
5.4.4	Data Tables .....	108
5.4.5	Workflow Templates .....	110
5.4.6	Activity Matching .....	111
5.4.7	Workflow Validation .....	116
5.4.8	Concretisation and Data Conversion .....	118
5.4.9	Workflow XML Format .....	120
5.4.10	Ontology Migration .....	122
5.4.11	Workflow UML Visualisation .....	123
5.5	Web Ontology Language Support .....	128
5.5.1	Label Types in OWL .....	128
5.5.2	Workflow Labels in OWL .....	131
5.5.3	Discussion and Future Use of OWL .....	133
5.6	Algorithm Complexity and Performance .....	135
5.7	Summary .....	136
6	Experiments, Use Cases, and Evaluation .....	138
6.1	Evaluation and Extension of the Ontology for Document Image Analysis .....	139

6.1.1	Building an Activity Repository .....	140
6.1.2	Discussion .....	144
6.1.3	Extension of Ontology .....	146
6.2	Evaluation of Search Functionality in Repositories .....	147
6.2.1	Activity Repository Decision Tree .....	151
6.3	Evaluation of Workflow Composition Functionality .....	154
6.3.1	Workflow for Page Segmentation .....	154
6.3.2	Replacing an Activity .....	157
6.3.3	Automated and Assisted Workflow Concretisation .....	159
6.3.4	Evaluation of Workflow Validation .....	163
6.4	Summary .....	166
7	Case Study: Digitisation of Historical Census Data .....	168
7.1	Census 1961 Digitisation Feasibility Study .....	169
7.1.1	Dataset .....	170
7.1.2	Ground Truth .....	170
7.1.3	Digitisation Pipeline .....	171
7.1.4	Performance Evaluation .....	173
7.2	General Considerations for Applying the Workflow System .....	175
7.3	Workflow Template Creation .....	176
7.3.1	Form / Table Analysis and Recognition .....	176
7.3.2	Pre-Processing Method Selection .....	177
7.3.3	Table Template Creation .....	181
7.4	Finding Workflow Templates .....	181
7.5	Template Concretisation .....	184

7.6	Pre-Processing Activity Selection.....	188
7.7	Providing External Input Data.....	190
7.8	Adding a Pre-Processing Step.....	193
7.9	Finalising the Census Workflow.....	195
7.10	Discussion.....	197
7.11	Summary.....	199
8	Discussion and Conclusions.....	201
8.1	Fulfilment of Objectives.....	201
8.2	Key Contributions.....	204
8.3	Comparison to other Semantics-based Approaches.....	206
8.4	Limitations.....	208
8.5	Applications and Future Work.....	210
8.5.1	Ontology.....	210
8.5.2	Workflow Model.....	212
8.5.3	Workflow System.....	217
8.6	Concluding Remarks.....	221
	References.....	223
	Appendix A – Label type hierarchies of the developed ontology (document image analysis).....	235
	Appendix B – List of All Implemented Interfaces and Classes.....	241
	Sequence Diagram for Workflow System Use.....	241
	Extended UML diagram for ontology-related classes and interfaces.....	242
	Extended UML diagram for workflow repository-related classes and interfaces.....	243
	Extended UML diagram for matching-related classes and interfaces.....	244



Extended UML diagram for workflow-related classes and interfaces .....	245
Extended UML diagram for data-related classes and interfaces .....	246
Alphabetic List of Classes and Interfaces.....	246
Appendix C – Software tools used for activity repository.....	255
Appendix D – Publication.....	266

## Acknowledgements

I thank Prof Apostolos Antonacopoulos for his guidance and help. I thank Dr Allen Fairchild for his valuable feedback. Furthermore, I want to thank my family, friends, and colleagues for their moral support.

## List of Terms and Abbreviations

<b>Activity / Workflow Activity</b>	A processing node in a workflow. Typically with data inputs and outputs [1]
<b>Cloud / Cloud Computing</b>	[2]
<b>Concretisation</b>	Making an abstract workflow non-abstract (concrete, executable) [1]
<b>DAG</b>	Directed Acyclic Graph
<b>Grid / Grid Computing</b>	Distributed computing resources [1]
<b>Label Type</b>	Element of the proposed semantic model / ontology
<b>METHONTOLOGY</b>	A methodology to design and maintain ontologies [3]
<b>OCR</b>	Optical Character Recognition
<b>Ontology</b>	A semantic model, usually for a specific domain
<b>OWL</b>	Web Ontology Language [4]
<b>PRImA</b>	Pattern Recognition and Image Analysis research lab
<b>RDF</b>	Resource Description Framework (Semantic Web)
<b>Repository / Workflow Repository</b>	System for storing and serving workflows
<b>SOAP</b>	Simple Object Access Protocol
<b>Template / Workflow Template</b>	A workflow with one or more abstract activities. Not executable Not executable
<b>UML</b>	Unified Modelling Language
<b>Workflow</b>	An encapsulation of activities (with control flow and dataflow) to model a software process.
<b>WSDL</b>	Web Services Description Language

## List of Tables

Table 1 - Comparison of three workflow systems .....	18
Table 2 - Root ontology terms for workflow activities and data objects.....	61
Table 3 - Advantages and disadvantages of different workflow base models.....	67
Table 4 - Mapping of XML format aspects for ontologies (dedicated vs OWL2) .....	128
Table 5 - Mapping of XML format aspects for semantic workflow data (dedicated vs OWL2).....	131
Table 6 - Runtime measurements for the workflow system prototype.....	136
Table 7 – Example activities (by category and in alphabetic order) .....	143
Table 8 - Results of workflow search experiment .....	150
Table 9 - Label types sorted by information gain (20 best and 20 worst ranked label types) .....	153
Table 10 – Labels used for the example workflow (limited to objects relevant to the case study) .....	156
Table 11 - Semantic labels of form or table analysis and recognition activity .....	177
Table 12 - Selected semantic labels for Image Pre-Processing Selection workflow template .....	180
Table 13 - Matching results for table recognition template.....	188
Table 14 - Overview of Census digitisation workflow creation.....	199

## List of Figures

Figure 1 - The four industrial revolutions (Christoph Roser at AllAboutLean.com) .....	3
Figure 2 - Grid and cloud computing (see [2]) .....	14
Figure 3 - Taverna Workbench ( <a href="http://dev.mygrid.org.uk/wiki">http://dev.mygrid.org.uk/wiki</a> ) .....	19
Figure 4 - Large Taverna workflow ( <a href="http://www.myexperiment.org/workflows/1198.html">www.myexperiment.org/workflows/1198.html</a> ) ..	21
Figure 5 - The Kepler workflow system ( <a href="https://code.kepler-project.org">https://code.kepler-project.org</a> ) .....	22
Figure 6 - The Triana system ( <a href="http://www.trianacode.org/">http://www.trianacode.org/</a> ) .....	23
Figure 7 - ASKALON user interface ( <a href="http://www.askalon.org/">http://www.askalon.org/</a> ) .....	25
Figure 8 - WINGS workflow composition system ( <a href="http://www.wings-workflows.org">www.wings-workflows.org</a> ) .....	32
Figure 9 - Operation discovery (adopted from Gubala and Bubak) .....	33
Figure 10 - Upper ontology in ASKALON (see [1]) .....	35
Figure 11 - Example ontology for ASKALON (partial). Meteorology. (after Qin and Fahringer) .....	36
Figure 12 - Service composition framework for the semantic grid (see [67]) .....	38
Figure 13 - Taverna workflow for comparing Optical Character Recognition (OCR) engines ([27]) .....	41
Figure 14 - States and activities of METHONTOLOGY (adopted from [3]) .....	48
Figure 15 - How the ontology grows: a) Fixed / waterfall; b) Basic incremental; c) Evolving prototypes (adopted from [3]) .....	48
Figure 16 - Expressivity, algorithm complexity, and ease of use of ontologies and systems using them .....	49
Figure 17 - Ontology representation levels (adapted from Obrst et al. [84]) .....	50
Figure 18 - Digitisation scenario .....	52
Figure 19 - Typical sequence of steps for document image analysis (adapted from [86]) .....	53
Figure 20 - Evaluation workflow of Europeana Newspapers Project [91] .....	57
Figure 21 - Initial term cloud for conceptualisation .....	60
Figure 22 - Refined term cloud for workflow activities .....	62

Figure 23 – Activity-related ontology terms.....	63
Figure 24 - UML notation of an activity. The activity is represented by a rounded rectangle. Input and output parameters are represented by rectangles on the border of the activity (inputs left or top; outputs right or bottom).....	69
Figure 25 - UML loop notations (left: generic form; right: specific form) .....	70
Figure 26 - Nesting of activities in UML (sequences denoted by connected data ports) 71	
Figure 27 - Decision nodes in UML activity diagrams (left: with fork and merge; right: structured node) .....	72
Figure 28 - Model of a data table.....	74
Figure 29 – Example dataflow for two activities.....	75
Figure 30 - A workflow template (left) and instantiated (concretised) workflows (middle and right) .....	76
Figure 31 - Illustration of semantic annotation via labels (labels can be attached to data ports and to the activity itself) .....	77
Figure 32 - Example for different views on a workflow system .....	78
Figure 33 - User groups in the example ontology .....	78
Figure 34 - Concept of activity matching .....	80
Figure 35 - Data conversion (UML) (Left: before adding conversion step; Right: after adding conversion step).....	83
Figure 36 - Architecture of prototype (rectangles: standalone software tools; circles: central objects).....	88
Figure 37 - UML Diagram for repository, workflow, and activity.....	89
Figure 38 - UML diagram for ontology and labels.....	90
Figure 39 - Ontology editor .....	91
Figure 40 - Ontology migration rules dialogue .....	93
Figure 41 - UML diagram for workflow repository and filter.....	95
Figure 42 - Repository Hub .....	96
Figure 43 - Workflow search interface .....	97
Figure 44 - Searching for data tables within a repository .....	99

Figure 45 - Workflow Editor (left: tree view of activities; right: details of selected activity)	100
Figure 46 - UML class diagram for data ports and data objects	101
Figure 47 - Data port dialogue	101
Figure 48 - Data type selection	102
Figure 49 - Selection of input port source	103
Figure 50 - Details of a concrete atomic activity (method name and version are provided and “Abstract activity” is not checked)	104
Figure 51 - Details of a “for loop” activity (special loop ports at the bottom right)	105
Figure 52 – UML class diagram containing loop port	105
Figure 53 - Details of Directed Acyclic Graph activity in Workflow Editor	106
Figure 54 - Details of If-Else activity in Workflow Editor (bottom: branches and conditions enlarged)	107
Figure 55 - Dialogue for adding a semantic label (here: label for a data port)	108
Figure 56 - UML class diagram for data tables	109
Figure 57 - Table columns as data source	109
Figure 58 - Data tables in the Workflow Editor	110
Figure 59 - Editing the content of data tables	110
Figure 60 - Workflow properties	111
Figure 61 - UML class diagram for matchers	112
Figure 62 - Matching for replacing an existing activity	114
Figure 63 - Details of match result	114
Figure 64 - Port alignment and metadata specification for replacing an activity	115
Figure 65 - Port alignment: Left: well aligned; Right: misaligned	115
Figure 66 - UML class diagram for workflow validation	116
Figure 67 - Validation dialogue	117
Figure 68 – Workflow concretisation	119
Figure 69 - Data type mismatch workflow validation error	120
Figure 70 - Flow chart for label type migration	122
Figure 71 - Ontology / label migration messages	123

Figure 72 - UML-like workflow visualisation of example with directed graph activity (top: entire workflow; bottom: left part enlarged) .....	124
Figure 73 - UML-like workflow visualisation of example with if-else activity.....	125
Figure 74 - UML-like workflow visualisation of example with for-loop activity.....	126
Figure 75 - UML-like workflow visualisation of complex example (bottom: enlarged left part).....	127
Figure 76 - Label ontology as visualised in Protégé.....	130
Figure 77 - Semantic workflow data as visualised in Protégé.....	132
Figure 78 - Example of an atomic activity in UML notation (including labels for data ports and activity) .....	141
Figure 79 - Search filters in Repository Hub (root label types in bold; child label types indented according to depth in hierarchy; numbers reflect the number of labels found within the current repository) .....	148
Figure 80 - User interaction with search interface.....	149
Figure 81 - Decision tree creation (left) and information gain ranking (right) in WEKA software tool .....	152
Figure 82 - Decision tree for activity labels (partial).....	153
Figure 83 - Example workflow template with a for-loop and directed graph activity (UML notation).....	155
Figure 84 - Activity matching result with details (missing label for "automated" highlighted).....	158
Figure 85 - Pre-processing activity of workflow template replaced with "Border Removal" activity .....	159
Figure 86 - Setup of workflow concretisation .....	160
Figure 87 - Result of automated workflow concretisation (top: three concrete atomic activities; bottom: concretisation log) .....	161
Figure 88 - Interactive workflow concretisation.....	162
Figure 89 - Selected validation messages for the segmentation example workflow (a: data type mismatch; b: label mismatch; c: abstractness; d: missing description; e: missing data type).....	165

Figure 90 - Cycle detection (left: directed graph activity with cycle; right: validation error message) .....	166
Figure 91 - Examples of Census 1961 scans .....	170
Figure 92 - Aletheia Document Analysis System.....	171
Figure 93 – Census digitisation pipeline.....	172
Figure 94 - Input image (left) and FineReader OCR result (right; visualised with Aletheia software tool).....	173
Figure 95 - Illustration of template matching (the goal is to find the best matching position of the template (red) with the page content (green)) .....	173
Figure 96 - Processing and evaluation framework .....	174
Figure 97 - Visualisation of evaluation result within the PRIMa Layout Evaluation tool .....	175
Figure 98 - Workflow template for form or table recognition (UML notation) .....	176
Figure 99 - Workflow for conditional image pre-processing (UML notation).....	178
Figure 100 - Workflow template for image pre-processing selection; top: overview, bottom: enlarged inner for loop (UML, simplified) .....	179
Figure 101 - Image pre-processing method selection template in Workflow Editor.....	180
Figure 102 - Table template creation workflow (UML).....	181
Figure 103 - Workflow discovery by content of interest.....	182
Figure 104 - Workflow discovery by activity domain.....	183
Figure 105 - Partial concretisation of table recognition template (top: result of automated process; bottom: result of assisted process).....	185
Figure 106 - Adding a new activity via text search .....	187
Figure 107 - Unconnected input ports in pre-processing selection workflow .....	190
Figure 108 - Executable image pre-processing selection workflow; top: whole workflow, bottom: enlarged inner for loop (changes made to template highlighted).....	192
Figure 109 - Validation error for data cardinality mismatch .....	195
Figure 110 - Adding a loop to resolve data cardinality mismatch (top: cardinality mismatch; bottom: solution using a for-loop activity) .....	196
Figure 111 - Data table that acts as source and target of activity data .....	214



Figure 112 - Data table used as internal state for finding the maximum of values .....	215
Figure 113 - Performance comparison workflow without (top) and with (bottom) activity references.....	217

# 1 Introduction

In addition to a general introduction of the subject matter, this chapter provides the motivation, objectives, main contributions, related publications, and arrangement of the PhD Thesis.

Scientific experiments, like any other processes, involve a collection of subtasks and data. Given the tasks are executed in a valid, predetermined sequence and the data is passed through appropriately, a meaningful result is produced. If the data is digital, then such processes can be automated and used repeatedly.

Any computer program can represent an automation of such nature, but specifically modelling scientific *experiments* as software processes provides several benefits: flexibility, understandability, transparency, ease of use, efficiency, fault tolerance, and data provenance. Some of these aspects can be achieved by using software scripts – a high-level form of programming using scripting languages; but only *workflow systems* offer solutions for all aspects. These systems represent an even higher level of abstraction in programming than scripts and typically include advanced features like graphical composition of workflows and distributed computing.

Nevertheless, workflow systems are not yet widely used within the scientific community (see [5]). One reason is that the state-of-the-art workflow tools still require a deep technical knowledge of the system and often programming skills. Another drawback is that workflow components (subtasks and data) can be combined in so many different ways and are so numerous that creating an effective and efficient workflow is a major challenge. The goal of this PhD project was to research if and how the addition of semantic data can help to overcome these issues.

The next section provides an overview of automation in general and its connection to workflow systems.

## 1.1 Processes and Automation

Scientific experiments, production processing pipelines<sup>1</sup>, or business workflows in general are all *processes* in the sense that they are composed of “a series of actions or steps taken in order to achieve a particular end”, as defined by the Oxford Dictionary of English [6]. *Automation* is therein described as “the use or introduction of automatic equipment in a manufacturing or other process or facility”. The definitions show that the two concepts are closely linked. In order to automate something, it first must be described as a sequence of activities. They also show the focus on manufacturing due to the historical origin of the idea of automation.

In computer science, but also in other disciplines, an experiment is usually a composition of software methods that process certain input data and produce a result that gives insight into the methods or the data. A production processing pipeline is very similar, with the main difference that a major emphasis is placed on efficiency (e.g. speed), robustness, and possibly flexibility.

*Scientific workflow systems* are the instrument of choice to formally describe experiments (or pipelines) and execute them. They represent a more dataflow-oriented specialisation of workflow systems.

There are parallels to industrial production processes which can be used to illustrate the developments in software process automation (see Figure 1). The first revolution can be compared to the emergence of computer programs which were executed independently. Data was managed mostly manually. The assembly lines of the second revolution are similar to scripting approaches, where fixed concatenations of programs can be called repeatedly. The third revolution has common features to the use of workflow systems in computing. Both offer more flexibility, robustness, and quality.

---

<sup>1</sup> We will use the term *pipeline* or *processing pipeline* for well-defined software processes that take a specific type of data as input and produce a specific output. Although a pipeline is a workflow, we use the term pipeline where we want to distinguish the *concept* of the software process from an actual workflow representation created with a workflow system.

The fourth revolution is happening in today's industry and is the most interesting one for this work. It changes the industrial landscape by incorporating cloud computing, the Internet of Things, and general-purpose robotics. This can only be achieved if the overall system has information on and control over each individual component (machinery and product parts). As a result, assembly lines can be self-organising and very flexible. Now there is automation in the setup of the production process itself (which is an automation of the production).

Similarly, to achieve automation in software workflow creation, each individual part (software tools and data items) must be enriched by (semantic) data in order to enable a composition system to make decisions and create valid workflows.

The next section introduces the domain of document image analysis and describes the significance to this PhD research.

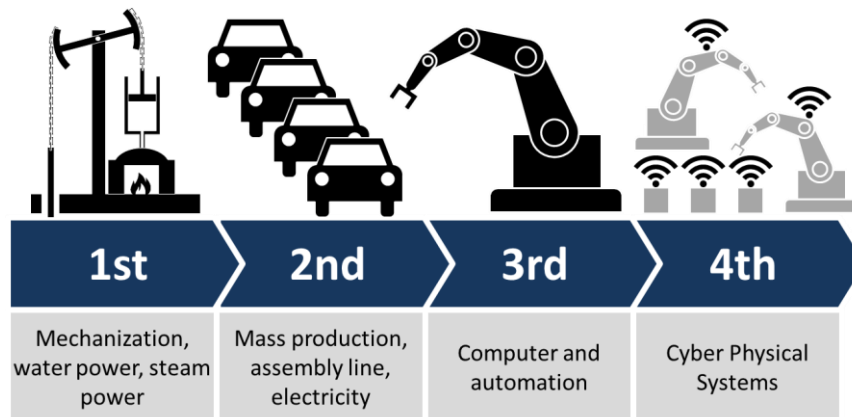


Figure 1 - The four industrial revolutions (Christoph Roser at AllAboutLean.com)

## 1.2 Relation to Document Image Analysis

*Document image analysis (DIA)* aims at automatic interpretation of document images (e.g. scans of printed books) [7]. Since all data is typically available in digital form, the field is most suited for modelling processes as scientific workflows. Nevertheless, based on what is reported in the literature, researchers in document image analysis only recently started to show interest in using workflows. In contrast, within the domains of astrophysics and

bioinformatics process modelling has been used for many years, possibly because those are traditionally data-rich and processing intensive fields.

The increased interest in workflows within DIA is likely driven by the rapidly growing volume and type of scanned documents such as books, archival prints, and newspapers, but also by the increase in available processing methods and data formats (due to research and development).

Typical tasks include the creation of experimental digitisation pipelines to recognise the layout of document images (i.e. scanned pages), to automatically transcribe the text content (Optical Character Recognition - OCR), to post-process (e.g. text post-correction), to perform high-level content understanding, and to evaluate the performance of methods. The term *digitisation* is commonly used to relate to the image acquisition (scanning) and subsequent processing steps.

To achieve those tasks, there is a plethora of tools and methods featuring:

- Different target platforms (operating system or other prerequisites for installed software).
- Different input/output formats (various image file formats, page content representation formats, text encoding variations etc.)
- Specific strengths/weaknesses and constraints (e.g. tools that specifically target historical data).
- Different levels of required settings (no settings, default settings, or context-specific settings).

Even for a basic digitisation pipeline, the options to consider are numerous: variations of pre-processing steps (enhancement, binarisation etc.), page analysis and recognition software and its parameters, input image formats, output formats with different features, and possibly post-processing methods. For experiments or feasibility studies, performance evaluation methods must be considered as well.

There is a growth in digitised printed material and there are numerous unsolved DIA challenges. Marc Wilhelm Kuster confirms this in his keynote for the 2011 International Conference on Document Analysis and Recognition [8] and adds that most of humanity's printed material is yet to be digitised. Le Bourgeois et al. [9] provide an extensive overview

of challenges in digitisation and digital libraries. They stress the need for collaboration between libraries and the research community.

In real-world situations, users in libraries, archives, and other institutions involved in digitisation of documents often use an arbitrary combination of tools to complete their tasks (commonly without knowing the limitations). One possible reason for this is the aforementioned ‘status quo’ of too many and too different tools and data formats. There is not enough easy-to-access information to make a sophisticated decision (it is easier to just use what is known, even if the quality of the results might not be the best). Even experts tend to use standard methods because setting up more complex experiments is too labour-intensive. This also makes it difficult to establish a newly developed method for a certain problem in the community, even if it performs better than a standard approach. Furthermore, there is no one-fits-all method. Commercial text recognition systems, for instance, perform well on different types of documents, but they are optimised for robustness and wide applicability, not to maximise the recognition quality for niche material (which might be the target data).

More details on document image analysis are provided in Chapter 3. The next section formalises the motivation for the PhD project.

### 1.3 Motivation

It lies in the nature of scientific domains to accumulate new approaches and solutions to problems. The emergence of the Internet led to an acceleration of available software tools, data collections and formats, and published processes. It is this vastness of possible component combinations that poses a major problem for researchers and other users. For each problem, there are usually several alternative methods that can be applied, each restricted to different inputs, producing outputs in different formats, and requiring specific settings or training. It is very difficult, even for experts, to keep track of developments and build up or maintain the required knowledge to select the most suitable tools for a task at hand.

Scientific workflow systems are a step in the right direction, but, to date, creating workflows is a challenging task requiring various skills and deep knowledge of the used system. For non-experts, it is virtually impossible to understand or design workflows other than trivial ones. Existing workflow systems generally require software development experience to be able to use them in a productive way. There is little automation or assistance.

An easy-to-use workflow composition system can be the answer. Given a high level of automation, workflow creation can become more accepted and used. In combination with searchable workflow repositories, scientific methods (software tools or proven workflows) would be more visible and, as a result, users could discover them using the properties they require for a specific use case (source data, target data etc.).

Most of the information on existing software, data, and formats only exists in written form, intended for human consumption. One main aspect of this PhD project was to research the use of machine-processable semantic information to enhance scientific workflow systems. This and the general objectives are described in the next section.

## 1.4 Objectives and Methodology

The aims of the PhD research were to investigate automation approaches for scientific workflows and develop strategies and solutions to simplify the creation and use of workflows for experiments and processing pipelines in a representative target domain (document image analysis and recognition). Or in other words, to prove or disprove the following hypothesis:

*By creating a formal model for document image analysis, by semantically annotating components and data, and by using this information algorithmically within a workflow system it is possible to automate common tasks (within such system) and assist users in workflow creation and management.*

It should be stressed that, while workflows themselves represent the automation of a process, this PhD project focusses on the automation of the *creation* of workflows.

The objectives are as follows:

1. To devise a modelling approach for representing semantic knowledge suitable for use in workflow systems.
2. To develop a semantic model for the domain of document image analysis, covering aspects relevant for workflow design and management.
3. To develop algorithms that use semantic information to support the user in workflow-related tasks (i.e. discover methods for more automation/support, for instance in workflow design).
4. To create a framework that enables to create, manage and use semantic information in combination with the above algorithms and general workflow-related functionality.
5. To test the model, the algorithms, and the framework on real-world data from the domain of document image analysis.

The research was carried out by:

- Determining a suitable approach to model semantic information that is expressive enough and can be applied to workflow components.
- Creating a semantic model for document image analysis using an ontology engineering approach.
- Developing a prototype of a workflow composition and management system that incorporates the semantic model as well as features for assisted and automated workflow composition (using semantic data).
- Proving the validity of the approach by applying it to real-world use cases and evaluating impact, advantages and disadvantages in detail.

By following the above methodology, a number of contributions to knowledge were achieved, the main of which are outlined next section.



## 1.5 Main Contributions

The key contributions of this work can be summarised as follows:

- A new approach for incorporating semantic information into scientific workflow systems: a label-based layer of data adds knowledge on the meaning of workflow components (subtasks/activities and data).
- A semantic model for workflow components in the domain of document image analysis and recognition: an ontology for the target domain created using a standard methodology for ontology engineering.
- A workflow system prototype: a fully-functional workflow composition tool and a workflow repository including the proposed semantic layer.
- Algorithms for (semi-)automation of workflow composition and discovery: new algorithms using a matching approach to make use of the semantic information and add assistive features to the workflow system.
- Proposed worked-out solutions to real-world use cases using the semantics-enriched workflow system, showing the impact of the assistive features.

The main outcome of the PhD research can be seen as the foundations of a complete future workflow system that, among others, enables non-domain experts to create valid workflows and researchers to conduct more complex experiments.

The next section lists publications by the author with direct or tangential relation to this PhD research.

## 1.6 Publications

Below are peer-reviewed publications by the author related to this work.

### *Directly Related*

(1) “Ontology and Framework for Semantic Labelling of Document Data and Software Methods”, C. Clausner, A. Antonacopoulos, Proceedings of 13th IAPR International

Workshop on Document Analysis Systems (DAS2018), Vienna, Austria, April 24-27, 2018, pp. 73-78.

Peer-reviewed publication for selected topics from the PhD project. The paper was selected for oral presentation at the DAS2018 workshop in Vienna, Austria.

(2) “Creating a Complete Workflow for Digitising Historical Census Documents: Considerations and Evaluation”, C. Clausner, J. Hayes, A. Antonacopoulos, S. Pletschacher, Proceedings of the 2017 Workshop on Historical Document Imaging and Processing (HIP2017), Kyoto, Japan, November 2017, pp. 83-88.

Digitisation workflow used as case study in Chapter 7.

(3) “Unearthing the Recent Past: Digitising and Understanding Statistical Information from Census Tables”, C. Clausner, J. Hayes, A. Antonacopoulos, S. Pletschacher, Proceedings of Second International Conference on Digital Access to Textual Cultural Heritage (DATECH 2017), Goettingen, Germany, 01 - 02 June 2017.

More information regarding Chapter 7.

### ***Informed from***

Competition papers for the International Conference on Document Analysis and Recognition (ICDAR) (2011 - 2017)

- “Historical Document Layout Analysis Competition”, A. Antonacopoulos, C. Clausner, S. Pletschacher, C. Papadopoulos, ICDAR2011, Beijing, China, September 2011.
- “Competition on Historical Newspaper Layout Analysis”, A. Antonacopoulos, C. Clausner, S. Pletschacher, C. Papadopoulos, ICDAR2013, Washington DC, USA, August 2013.
- “Competition on Historical Book Recognition”, A. Antonacopoulos, C. Clausner, C. Papadopoulos, S. Pletschacher, ICDAR2013, Washington DC, USA, August 2013.

- “Competition on Recognition of Documents with Complex Layouts”, A. Antonacopoulos, C. Clausner, C. Papadopoulos, S. Pletschacher, ICDAR2015, Nancy, France, August 2015.
- “Competition on Recognition of Documents with Complex Layouts”, C. Clausner, A. Antonacopoulos, S. Pletschacher, ICDAR2017, Kyoto, Japan, November 2017.

A series of competitions in layout analysis and end-to-end page analysis and content recognition, showing the author’s experience in creating workflows using the concepts of the PhD research.

“Aletheia - An Advanced Document Layout and Text Ground-Truthing System for Production Environments”, C. Clausner, S. Pletschacher, A. Antonacopoulos, Proceedings of the 11th International Conference on Document Analysis and Recognition (ICDAR2011), Beijing, China, September 2011, pp. 48-52.

Aletheia is now a complete document analysis system with commercial success, showing the author’s experience in software development in the target domain.

“Scenario Driven In-Depth Performance Evaluation of Document Layout Analysis Methods”, C. Clausner, S. Pletschacher, A. Antonacopoulos, Proceedings of the 11th International Conference on Document Analysis and Recognition (ICDAR2011), Beijing, China, September 2011, pp. 1404-1408.

A flexible evaluation approach for page layout analysis methods. This work demonstrates landscape of different use cases and available methods, leading to a scenario-driven approach using evaluation profiles.

“A robust hybrid approach for text line segmentation in historical documents“, C. Clausner, A. Antonacopoulos, S. Pletschacher, Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, November 11-15, 2012, IEEE-CS Press, pp. 335-338.

A segmentation method developed within a major EU-funded project, showing the author’s experience in developing and applying components for digitisation pipelines.

“The IMPACT Dataset of Historical Document Images”, C. Papadopoulos, S. Pletschacher, C. Clausner, A. Antonacopoulos, Proceedings of the 2013 Workshop on Historical Document Imaging and Processing (HIP2013), Washington DC, USA, August 2013, pp. 123-130.

Publication about large dataset for historical printed material, showing the author’s experience in data collection and metadata provision.

“The ENP Image and Ground Truth Dataset of Historical Newspapers”, C. Clausner, C. Papadopoulos, S. Pletschacher, A. Antonacopoulos, Proceedings of the 13th International Conference on Document Analysis and Recognition (ICDAR2015), Nancy, France, August 2015, pp. 931-935.

Publication about large dataset for historical printed material, showing the author’s experience in data collection and metadata provision.

“Europeana Newspapers OCR Workflow Evaluation”, S. Pletschacher, C. Clausner, A. Antonacopoulos, Proceedings of the 2015 Workshop on Historical Document Imaging and Processing (HIP2015), Nancy, France, August 2015, pp. 39-46.

Evaluation of complete, large-scale OCR workflow, representing typical use cases in the target domain.

“Quality Prediction System for Large-Scale Digitisation Workflows”, C. Clausner, S. Pletschacher, A. Antonacopoulos, Proceedings of the 12th IAPR International Workshop on Document Analysis Systems (DAS2016), Santorini, Greece, April 11-14, 2016.

Prediction of OCR workflow performance (use case in DIA).

## 1.7 Thesis Overview

The remainder of this Thesis is organised as follows:

- **Chapter 2** (Automation and Scientific Workflows) introduces workflows and workflow systems, including a review of scientific workflow systems. This

chapter also describes existing approaches for workflow automation, some of which are based on semantic models.

- **Chapter 3** (A Semantic Model for Document Image Analysis and Recognition) concentrates on semantic models and explains the development of a new ontology for document image analysis workflows. In addition, the role of the Semantic Web in the context of scientific workflows is described.
- In **Chapter 4** (Designing a Semantics-Enriched Workflow System) a design for a new prototype workflow system is presented, which incorporates semantic features and automation approaches as a proof-of-concept for the methods and concepts that have been developed during this PhD research.
- **Chapter 5** (System Implementation) provides details on the implementation of the system design from Chapter 4.
- **Chapter 6** (Experiments, Use Cases, and Evaluation) makes use of the semantic model and the prototype to explore a number of use cases in the domain of document image analysis. The expressiveness of the proposed model is evaluated and it is explored how workflows can be composed with the system.
- **Chapter 7** (Case Study: Digitisation of Historical Census Data) describes how the workflow system can be applied to a real digitisation project.
- **Chapter 8** (Discussion and Conclusions) concludes the Thesis with a detailed discussion and suggestions for future work, including the use of the proposed concepts in other systems and workflow sharing platforms.
- The **appendices** contain the full ontology for document image analysis and recognition from Chapter 3, the list of classes of the developed prototype workflow system from Chapter 5, and a list of software tools employed in the use cases of Chapter 6.

## 1.8 Summary

Scientific workflow systems were introduced as a high-level modelling and execution approach for experiments and other processes. It was pointed out that current systems have

shortcomings in terms of automation and assistance, effectively hindering inexperienced users in creating valid workflows. This problem is intensified by the target domain's (and other domains') status quo of a vast and growing landscape of software tools, data collections, and data formats. This and the increasing need for solutions in the more specific domain of document image analysis were stated as the motivation for this PhD project.

The main objective was to research the use of a semantic layer in workflow systems, to achieve automation in workflow composition and management. The chapter concluded by listing the key contributions (approach for semantic layer, ontology for document image analysis, prototype workflow system, and case study) and related publications by the author.

The next chapter reviews workflows and workflow systems in general and analyses the use of semantics.

## 2 Automation and Scientific Workflows

This chapter provides theoretical background on workflows, Grid Computing, and automation approaches. Prominent state-of-the-art workflow systems are detailed and compared.

### 2.1 Workflows and Distributed Computing

As defined by the Workflow Management Coalition [10], a *workflow* is “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant (a resource; human or machine) to another for action, according to a set of procedural rules”. Fox and Gannon [11] refined this, defining a workflow as “the automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service”. The term *Grid* (Grid Computing, The Grid) originated from the idea of making computing as widespread, reliable, and easy-to-use as an electrical power grid [1]. It is a synonym for distributed computing environments. Today, *Cloud Computing* is often used as a “catch all” term, although there are distinctions, as Foster et al. [2] point out. Figure 2 provides an overview on their interpretation of distributed systems.

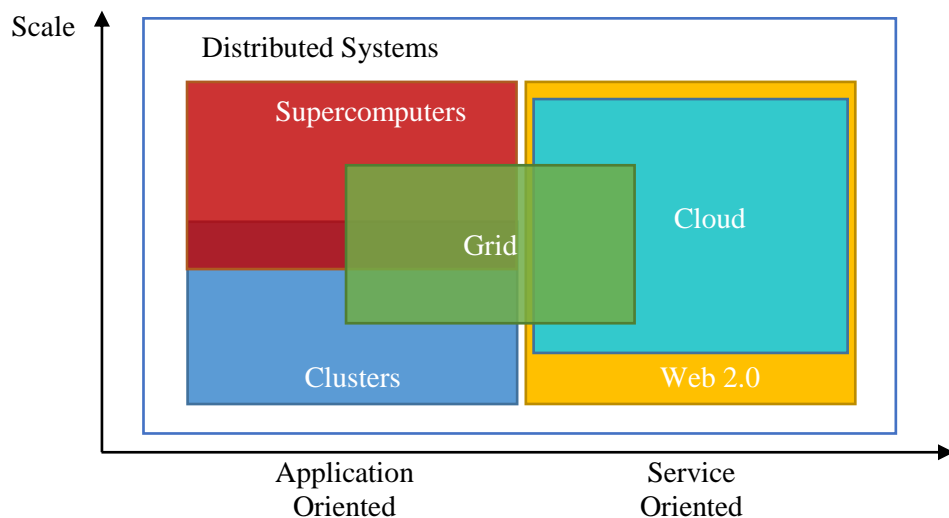


Figure 2 - Grid and cloud computing (see [2])

Originating from the business sector, workflows are increasingly adopted by the scientific community ([12, 13]). In the beginning this was mostly driven by research in physics and bioinformatics, where large amounts of data needed to be processed. The so-called *scientific workflows* have slightly different requirements in comparison to their counterparts from business process automation. Fahringer and Qin [1] describe scientific workflows as the “main programming model for the development of scientific applications on distributed systems”. They also state the three major properties as:

- (1) Execution on distributed systems where resources can vary.
- (2) Composition by scientists who are not necessarily programming experts.
- (3) Complex control and data flow requirements.

## 2.2 Workflow Systems

Workflow systems are software tools to create, modify, manage, and execute workflows. Yu and Buyya [14] propose a taxonomy for the classification of workflow systems, characterising them based on workflow design, information retrieval, workflow scheduling, fault tolerance, and data movement.

Advantages and disadvantages of using workflows (scientific workflows in particular) can be grouped into three categories, based on the paradigm they originate from.

### *(1) Characteristics of distributed systems:*

Distributed systems are usually heterogeneous with regard to computer systems (processing architecture, number / size of resources, operating system etc.) and networking (protocols, bandwidth etc.). This heterogeneity and the fact that often resources can join and leave the system at any time, lead to the requirement of adaptability, usually provided by a middleware layer. One major advantage, especially for large research projects, is scalability. The adaptable character of distributed systems allows the gradual growth, starting from only a few resources up to potentially millions of web services, data sources, processing nodes etc. [1].



Furthermore, distributing processing and data has the advantages of promoting collaboration between organisations and obtaining specific processing capabilities by spanning multiple administrative domains [14].

*(2) Scripted execution:*

Workflows represent a special form of scripted execution of processes. Scripting has a long tradition in research and was used before scientific workflow systems emerged. Advantages are reusability of scripts or part of scripts [15], repeatability of experiments, and automation in general [16].

*(3) Workflow system and graphical user interface:*

Workflow systems provide a high-level view of data and processing resources as well as of the control and data flow, making it easier to understand and modify workflows. Further common features are extended logging and monitoring, provenance management, failure tolerance, and sharing of workflows and/or result data. [5]

The service-oriented design (loosely coupled resources) of workflows provides a high degree of flexibility. Barker and Hemert [16] describe workflows as “the glue for distributed services, which are owned and maintained by multiple organisations”.

Graphical design tools allow the composition of workflows without the need to understand the underlying workflow scripting and modelling language [17].

Cohen-Boulakia and Leser [5] mention several negative issues with the current state-of-the-art. Foremost, they write that today’s systems are too complex for most domain scientists and offer little advantage over scripting. Further alleged problems are inadequate performance, difficult debugging, low reliability, incompatibilities (between data formats and invocation methods), and lack of standards.

The structure of workflow systems and distributed systems in general can be compared with the architecture of a conventional single computer [1]. The lowest layer (for a computer this is the hardware) is composed of the distributed resources such as servers, computer clusters, storage and network devices, and others. Fahringer and Qin call this the “Fabric”.

The next layer up is the middleware, with distributed operating system (equivalent to a conventional operating system such as Linux) and workflow systems as the programming environment (equivalent to an integrated development environment such as Microsoft Visual Studio). Workflows can be interpreted as an application or program and therefore represent the uppermost layer.

Yu and Buyya [14] propose a more complete model, building upon the model proposed by the Workflow Management Coalition [10]. They distinguish between processes for build time (workflow design and definition) and run time (workflow execution / control and interaction with Grid resources). At build time, Grid users create workflows using a Grid workflow application modelling and definition tool, producing a workflow specification. At run time, a grid workflow enactment service handles workflow scheduling, data movement, and fault management. It thereby interacts with the Grid middleware, which in turn handles Grid resources. Modelling tools and enactment services further interact with Grid information services holding resource and application information.

Several scientific workflow systems have been developed, often targeting very specific needs and research areas. Only a few however, have evolved to more complete and generic solutions. The most prominent of those systems are Taverna [17, 18], Kepler [19], Triana [20], and Pegasus [21, 22]. The following subsections provide details on the four frameworks, as well as another system called ASKALON [1], because it is of relevance for this work (used as basis for prototype). Table 1 provides a brief comparison. Curcin and Ghanem [23] provide an in-depth review of Taverna, Kepler, Triana, and others. Talia [12] also reviewed those systems (including Pegasus) and, in addition, mentions several open issues concerning workflow systems in general, including:

- Adaptive workflow execution models (to deal with elastic infrastructures).
- High-level tools for workflow composition (currently, the basic building blocks of workflows are simple and regular).
- Interoperability and openness (often proprietary data and ad hoc formats are used; standards are needed).

- Big Data management and knowledge discovery workflows (higher-level and scalable systems are required).
- Internet-wide distributed workflow execution (workflow systems are already distributed, but the scale is still limited).
- Cloud-based service-oriented workflows (more research needed for using this technology for heterogenous platforms).
- Exascale computing systems (research needed for massively parallel processes)
- Fault-tolerance (only few systems support this properly).
- Provenance and annotation (lack of management, visualisation, and evaluation of provenance data).

The Taverna, Kepler, Triana, Pegasus, and ASKALON systems are introduced next. All are candidates for adding the semantic features proposed in this thesis.

*Table 1 - Comparison of three workflow systems*

	<b>Taverna</b>	<b>Kepler</b>	<b>Triana</b>	<b>Pegasus</b>	<b>ASKALON</b>
<b>Availability</b>	Open source	Open source	Open source	Open source	Closed source, binaries available on request
<b>Activities</b>	Web services, Java snippets	Local programs, web services	Service tasks, control tasks	Execution task, data management	Web services, control flow
<b>Language</b>	Dedicated, XML-based format	Dedicated XML format MoML	Dedicated XML format for components	Dedicated XML-based format DAX	Dedicated XML format AGWL
<b>Nesting (sub-workflows)</b>	Yes	Yes	Yes, via task groups	Yes (sub-DAG)	Yes
<b>Implementation</b>	Java-based	Java-based	Java-based	Java and Python	Java-based
<b>Incarnations</b>	Desktop version, server version	Desktop version, various add-on modules	Desktop version, Triana service	Pegasus command line tool	Desktop version, runtime middleware services
<b>Active development</b>	Yes	Yes	No (last update 08/2014)	Yes	Unknown

## 2.2.1 Taverna

Taverna [17, 24-26] was originally developed by Oinn et al. at the University of Manchester for the purpose of conducting experiments in bioinformatics. It has also found widespread recognition in other fields, including document analysis (see for instance [27, 28]). It is solely based on web services, but local execution is possible through command line tool wrappers (that create local web services). Workflows are represented through a proprietary graph structure which can be stored using a dedicated XML-based format.

There is a very limited pool of standard types of processing nodes, but more complex workflows can be achieved by incorporating Java software snippets into the workflow (allowing for instance if-else conditions). Iteration (loops) are handled implicitly: if the input data of a processing node is a list of data items, they will be processed one after the other. The Eclipse<sup>2</sup>-based desktop version provides a graphical user interface with drag-and-drop features and online workflow repository integration (Figure 3). There is also a server-based version which can be accessed through a graphical web interface and web services.

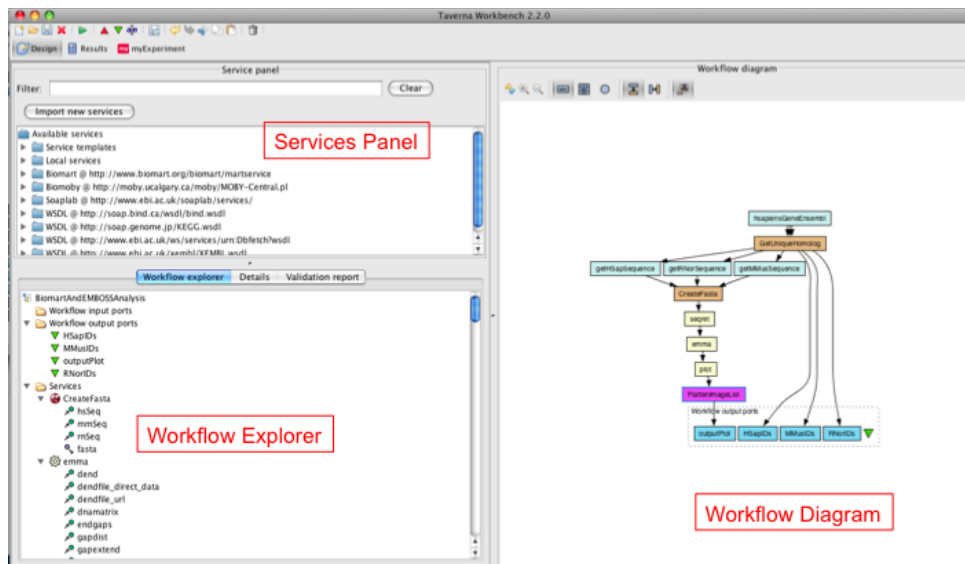


Figure 3 - Taverna Workbench (<http://dev.mygrid.org.uk/wiki>)

<sup>2</sup> [www.eclipse.org](http://www.eclipse.org)

Taverna provides a functional, data-centric programming approach, where many functions are handled implicitly (hidden from the user). This makes it easier to concentrate on “what to do” instead of describing “how to do it”, but it also leads to limitations (e.g. less control in processing loops) (see [29]).

In Taverna, the user is responsible for choices of resources and/or services. Workflows are therefore not flexible, with the only exception being the possibility of defining a set of optional services for one task (although the method of late binding is not clear) [1].

The graphical representation provides a good overview of a workflow but can be cumbersome to navigate for larger experiments (as can be seen in Figure 4, for example). Execution progress of a running workflow is conveniently reflected in the workflow graph through colour changes. Integrated sharing functions make reuse of workflows easier, although only keyword-based search is available. Currently, there are no constraints on how to connect workflow elements, which creates a major risk for workflow composition errors. Creating ad hoc workflows for quick experiments is not straightforward since all processors (processing nodes) need to be web services. This complicates the usage of local software tools. Oinn et al. [17] mention options for open source tool wrappers that create web services, but this solution still adds significant overhead to experiments and requires additional knowledge of software programming.

The workflow representation format has undergone several major changes over time, from SCUFL (Simple, Conceptual, Unified Flow Language) to “t2flow” and eventually to SCUFL2 for the latest version of Taverna.

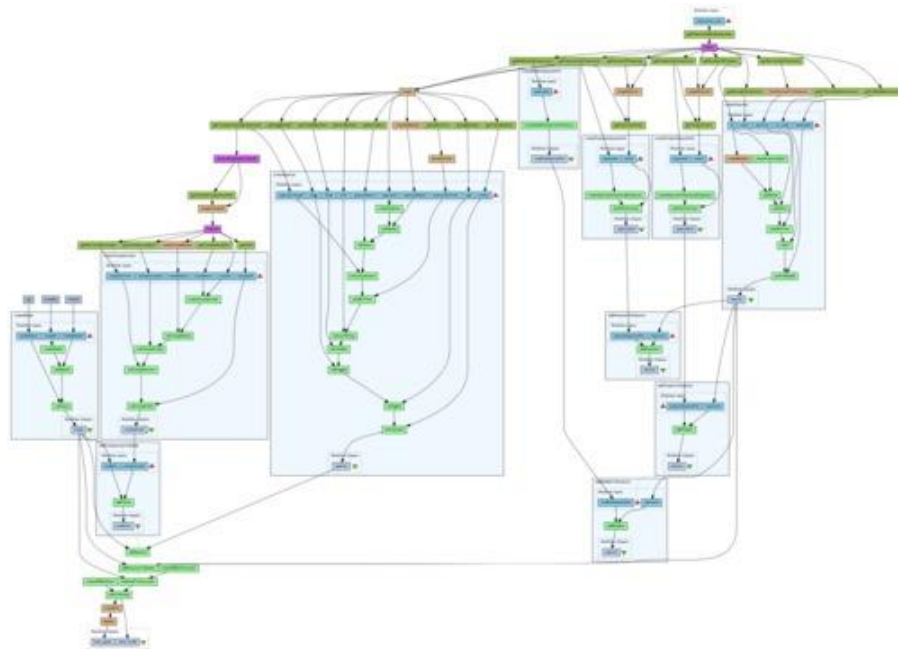


Figure 4 - Large Taverna workflow ([www.myexperiment.org/workflows/1198.html](http://www.myexperiment.org/workflows/1198.html))

### 2.2.2 Kepler

The Kepler system was created by Ludäscher et al. [19] as an extension of the Ptolemy II software [30], an actor-oriented design tool. Workflow components are therein represented by actors which represent operations or data sources [23]. Kepler extends Ptolemy II by adding a web service actor, allowing a wider range of scientific workflows.

An independent component of the Kepler system called *director* forms the execution model, which makes decisions when to schedule the execution of each actor. Kepler's design is influenced by its original focus on data analysis and modelling (e.g. physics ecosystems or bioinformatics web services). Dataflow is based on *tokens* which contain the data and are passed from actor to actor.

Figure 5 shows the Kepler desktop tool featuring a well-designed drag-and-drop graphical workflow composition interface [11].

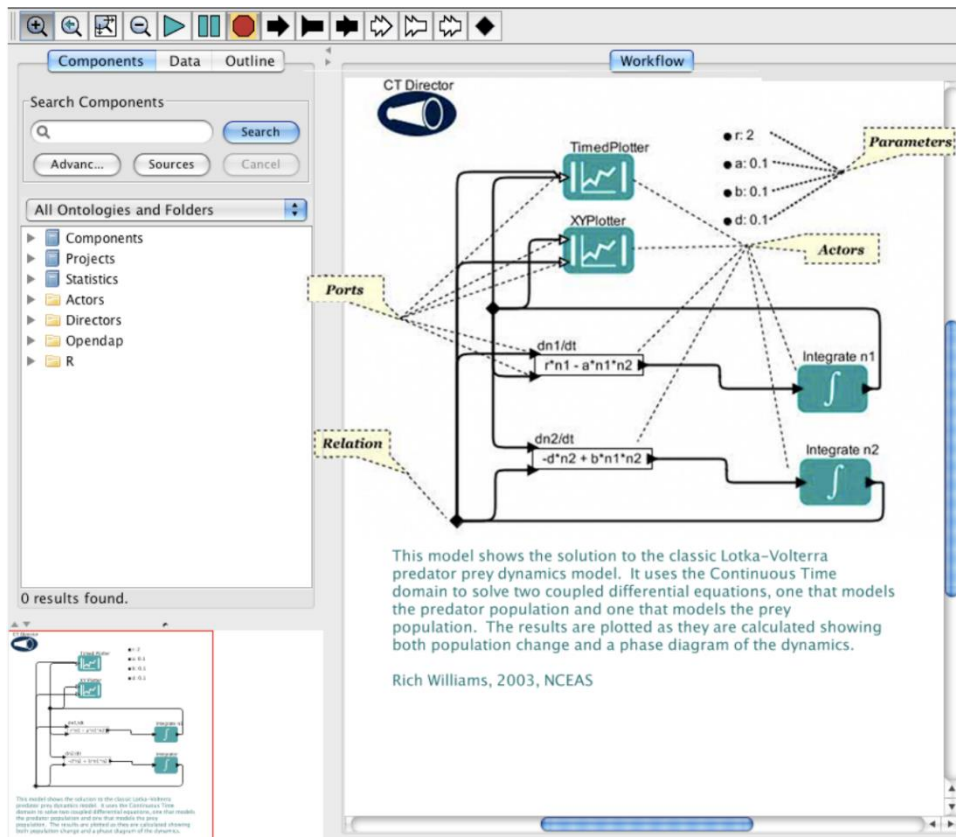


Figure 5 - The Kepler workflow system (<https://code.kepler-project.org>)

Workflows in Kepler are more static in the sense that the execution order of processes is predetermined and cannot be based on the outcome of a processing node [23]. Ludäscher et al. mention several research topics that need to be addressed further in Kepler [19]:

- Higher-order constructs for better control flow and less complex workflows
- Third-party data transfers to optimise dataflow between web services
- Detached execution (background execution for time-consuming workflows)
- Fault tolerance
- Data provenance
- Semantic links (in order to determine which actors and datasets fit together semantically)

The latter is described as a “hard problem” and ontologies are mentioned as possible solution for semantic type systems (see also [31]).

Abstract workflows (workflows that have placeholder components) are not supported in Kepler (i.e. only concrete workflows are supported). Also, domain scientists are required to have knowledge of low-level implementation technologies (e.g. for data movement) [1].

### 2.2.3 Triana

Triana [20, 32], created by Majithia et al., is a complete workflow environment including: service discovery methods, composition methods, transparent execution methods, and publishing of services. It originates from an astrophysics project for gravitational wave detection. It entails a graphical user interface (Problem Solving Environment – PSE, see Figure 6) with drag-and-drop functionality for *units* (activities) and *cables* (control flow and dataflow). Units can be web services, group tasks, or control tasks.

Triana has its own dedicated XML-based format for storing components (Taskgraph), but it also provides a reader component for the widely used business process description format BPEL4WS [33].

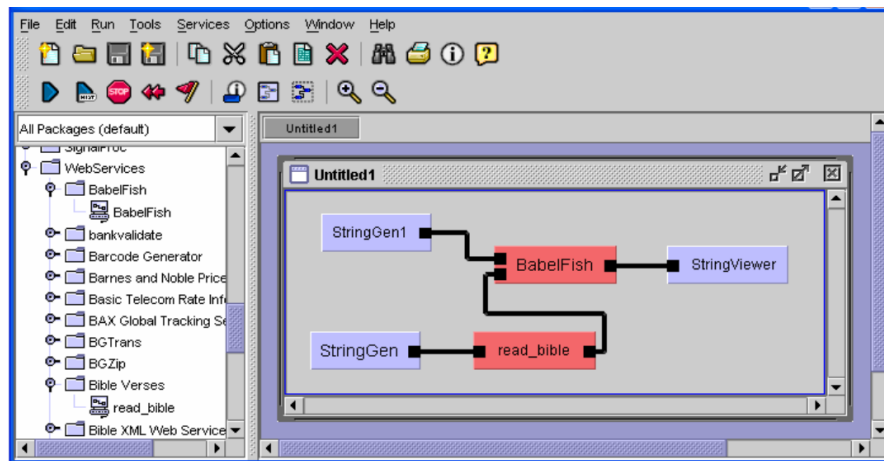


Figure 6 - The Triana system (<http://www.trianacode.org/>)

Processing nodes in Triana only allow one output port (not multiple ports with different data types, as possible in other systems) [23]. Component composition is made type-safe through attached data type information. One of Triana's strongest points is the mature user interface including GUI builders for interactive workflows [34].



Triana does not provide failure recovery as such (e.g. no retrying of a service) but it provides user feedback of failed workflow components [34]. The Triana project does not seem to be active any more, the last update to the Triana source code on GitHub [35] being in 2014.

By providing a high-level programming interface (GAT/GAP – Grid Application Toolkit and adapter API), Triana hides low-level implementation technologies from the user. However, this is not the case for hardware resources. Users have to make hardware-related decisions at design time [1].

#### 2.2.4 Pegasus

The Pegasus Workflow Management System [21, 22] (by the University of South California) is based on abstract workflows modelled by directed acyclic graphs. Nodes therein represent tasks and edges represent data and control flow. Abstract graphs are made executable (concrete) step by step using a mapping engine to allocate local or remote execution and data resources. This can extend the original graph by data management nodes.

Workflows are stored in a dedicated XML-based format called DAX (Directed Acyclic graphs in XML).

The Pegasus system has been used in several domains such as bioinformatics, oceanography, geology, and astronomy (see [12]). Recently, it was used for analysing gravitational waves from the LIGO (Laser Interferometer Gravitational-Wave Observatory) project.

Pegasus itself does not provide a graphical user interface. However, there are third-party systems that build upon Pegasus, providing graphical workflow composition (e.g. WINGS [36], see also subsection 2.5.1).

#### 2.2.5 ASKALON

The ASKALON system is comprised of a composition tool (Figure 7) and middleware execution and scheduling services. It was originally developed by Fahringer and Qin [1, 37, 38] from the University of Innsbruck. Workflows consist of activities (atomic / web services,

loop, if-else, sequence, graph) and data items. The composition is based on UML activity diagrams and partly based on previous work by Pillana, Qin, and Fahringer [39]. Dataflow is achieved through connecting input and output ports of activities.

Workflows are stored and shared using the system-specific AWDL format (Abstract Workflow Description Language).

ASKALON is closed source (source code not public), but executables are available on request. Workflows have a detailed and powerful control flow via explicit conditional (e.g. “if”) and loop activities. As a result, workflow composition closely resembles programming and is less intuitive than more abstract approaches (such as used in Taverna).

All of the presented systems use their own specific workflow language. The next section provides more details on workflow representation approaches in general.

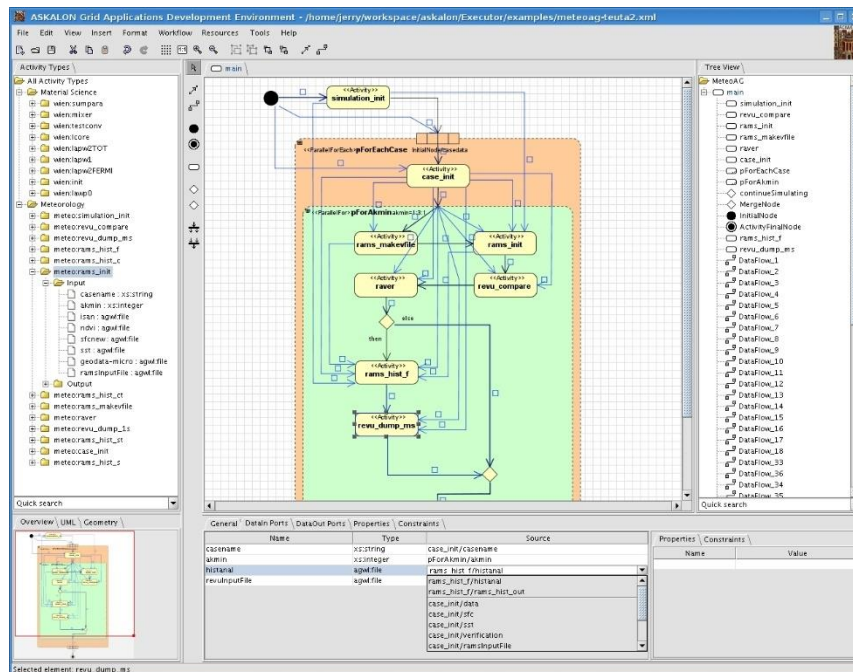


Figure 7 - ASKALON user interface (<http://www.askalon.org/>)

## 2.3 Workflow Languages and Modelling Paradigms

Almost every workflow system uses a proprietary workflow representation or language. In general, these languages are based on one of the following paradigms: Directed Acyclic Graphs (DAG) [13], Petri nets [40-42], or Event-Driven Process Chains (EPC) [43]. Nevertheless, some effort has been directed towards standardisation using UML activity diagrams [1] or the Business Process Execution Language (BPEL) [33] which is the quasi standard for defining business processes [44]. Juhnke et al. [45] propose a variation called SimpleBPEL with the goal of making BPEL more accessible to non-experts.

Other proposals include the development of new workflow languages that present a superset of all existing workflow representation approaches [23]. List and Korherr [46] propose a generic meta-model to compare and evaluate seven business process modelling languages, including EPC, UML activity diagrams, and Petri nets. Ivanova and Stromback [47] propose a general model (based on Open Provenance Model – OPM [48]) which combines features of popular workflow systems to enable independent querying of workflows. In a more recent work, Plankesteiner et al. [49] present a fine-grained interoperability solution for four workflow systems (ASKALON, MOTEUR, WS-PGRADE and Triana). They split the problem into abstract level (solved with an Interoperable Workflow Immediate Representation – IWIR – a bridging language) and concrete level (solved via bundles of IWIR representation and concrete task representations for workflow instantiation and execution).

For scientific workflow systems, there is no prevailing standard for workflow representation. Interoperability efforts have been made, but all require substantial conversion work and all have limitations when compared to the native workflow languages. While BPEL is successful in business workflows, it is not suitable as intermediate scientific workflow language [49].

Despite the many different modelling approaches, there are some common features:

- Atomic tasks (activities, actors, processing nodes, units, services): An algorithm or method executed by a computational unit. Atomic tasks can be seen as a black box with given inputs and expected outputs.

- Compound tasks (such as sub-workflows or sequences): Nesting for better reusability and understandability.
- Data: Input for and output from tasks or the whole workflow. Data objects have a data type and can be compounded in some cases (i.e. data collections)
- Data flow (data connections, links, cables, tokens): Specifies the way data flows from task to task.
- Control flow (conditions, loops, directors, control activities): Specifies the structure of a workflow, together with the data flow. Control flow steers the processing at execution time, reacting to the data at hand.

Another important point for interoperability is the sharing of created workflows. This is discussed next.

## 2.4 Workflow Repositories

To enable efficient sharing and reuse, workflow systems need to be complemented by workflow repositories. Out-of-the-box, most systems only allow to view and possibly reuse the workflows that have been previously created locally. A repository, on the other hand, is an independent database of existing workflows, allowing for searching, retrieval, and sharing. Repositories can be stand-alone or linked into the user interface of workflow systems. The latter allows direct import of existing workflows or components into local projects. Searches can typically be carried out via keywords or metadata (e.g. author).

Several workflow repositories have been proposed, some of them being used widely. One of the most successful is myExperiment [50, 51], a web-based solution, also featuring social aspects to comment on and discuss workflows as well as create user groups and experiments. The aforementioned system Taverna integrates with the myExperiment platform and allows to directly use publicly available workflows. At the time of writing, there are 2,137 Taverna workflows published on myExperiment (the oldest from 2007). Considering the long time period and the variety of research domains, this is not a very large number. Also, only 13 workflows were published in 2017 (see [www.myexperiment.org](http://www.myexperiment.org)).

Cohen-Boulakia and Leser [5] reviewed further repositories. They state that “major features and functionalities must be added to be truly useful to end-users”. Currently, searching for workflows can only be done using keywords and not the actual nature (components or input and output data) of the workflows.

Garijo et al. [15] mention myExperiment and CrowdLabs [52] as emerging repositories that made sharing easier. At the same time, they caution that reuse is still a major challenge because the workflows or fragments need to be fully understood to be used correctly. The authors say that this hurdle causes developers (researchers) to start workflows from scratch.

Web Service Discovery (WSD) is related to workflow repositories as both offer interfaces for discovering processing entities (with input and output data). In WSD, service providers can register and thereby publish a service in a Web Service Registry with the UDDI (Universal Description Discovery and Integration) standard [53], for example. The UDDI format is platform-independent and is designed to work in cooperation with other web service standards such as SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language).

UDDI contains three main components:

1. “White pages” for information about the provider (the business).
2. “Yellow pages” for a classification of the service using standard taxonomies.
3. “Green pages” for information about how to access the web service.

While this represents standardised semantic information, it is limited to a basic description of a web service as a whole, not its detailed functioning or components.

The next section provides an overview of existing uses of semantics in workflows and automation in workflow systems.

## 2.5 Automation and Semantic Features

Even though current workflow systems and repositories are certainly powerful tools, they lack a certain degree of automation and assistance (see [54]). When designing workflows

with Taverna for instance, the user needs to know exactly which processing nodes are required to reach a specific goal. Neither does Taverna give feedback on the data type compatibility of nodes nor does it suggest or add data conversion nodes where necessary. There seems to be a consensus that the usage of semantics is most promising to help to overcome these shortcomings.

The term *ontology* is often used in context of semantics and modelling. The Oxford Dictionary [6] defines an ontology as “A set of concepts and categories in a subject area or domain that shows their properties and the relations between them”. More information can be found in Chapter 3.

So far, little research has been conducted to model and exploit semantic data for scientific workflows. Often, concepts and standards for the Semantic Web are adopted. Gil et al. [55] propose a method based on workflow templates that allow for semantic constraints and other concepts (more in 2.5.1). They represent workflows using W3C’s Web Ontology Language (OWL) [4]. This concept has been refined by Hauder et al. [56], aiming at validating data mining workflows and helping users in the design process.

Gubala et al. [57] also make use of the OWL standard, integrating it with a Petri net based workflow model (a Petri net is a mathematical modelling solution for distributed systems). They try to achieve a tool that “understands the domain-specific information for an application and that can deal with semantics of the application’s elements” (discussed in 2.5.2).

Garijo et al. [15] analysed 177 workflows to identify common semantic denominators to provide abstract, high-level views in their system to improve understandability of workflows. As a result, they report data-oriented and workflow-oriented *motifs*. Data-oriented motifs include: data retrieval, data preparation, data movement, data cleaning, data analysis, and visualisation. Workflow-oriented motifs include: synchronous / asynchronous invocations, internal macros, human interactions, atomic workflows, workflow overloading, and composite workflows. The authors analyse the frequency of those motifs and propose a few good practices for workflow development (e.g. modularisation). The goal of an automatic abstraction of workflow components is only outlined and has not been realised.

Nadarajan, Chen-Burger, and Malone [58] outline a strategy for workflow composition in video processing using semantics in a design layer. They analysed the capabilities of Triana, Taverna, Kepler, and others with respect to video processing with semantic support. They conclude that none of the systems are satisfactory, stating that the incorporation of semantic technologies is too limited and not sufficiently integrated. Specifically, they argue, ontology handling and manipulation should be part of a system and not purely external (third-party tools). They proposed a theoretical framework with a design layer, a workflow layer, and a processing layer, but an implementation has not been realised.

Fahringer and Qin [1] and Siddiqui et al. [59] provide an extensive overview on semantic-based workflow composition, discussing ontology-based workflow representation, implementation approaches, and experimental evaluation. Fahringer and Qin incorporated semantic features into the ASKALON workflow system, details are provided in 2.5.3. Siddiqui et al. propose a rule-based semantic framework for automatic synthesis of complex (sequential / parallel) activities from basic activities. They demonstrate this on an example for video processing, but do not make clear how their approach can be easily transferred to other domains. The definition of semantic rules is a very complex task, requiring extensive knowledge of Semantic Web technologies and the Protégé software [60]. The authors mention that activity providers can annotate their activities with rules, but they do not discuss how non-experts can be enabled to do so.

Xing, Dikaiakos, and Sakellariou [61] propose an ontology for the grid infrastructure itself to allow reasoning on resources, middleware, services, applications, and users. They defined a core ontology including concepts for: grid middleware, grid components (storage, user interface etc.), grid applications, users, resources (computing, network etc.), and services (scheduler, monitor etc.). Although the ontology is still rather basic, it is a promising proposition. However, issues such as reasoning and ontology querying have not been addressed.

Automatic workflow composition and retrieval from repositories can be based on standard reasoning algorithms from the artificial intelligence domain as described by Gil et al. [62] for instance. This is discussed further in section 2.6.

The following subsections review several state-of-the-art approaches for automation in workflow systems based on semantic information.

### 2.5.1 Semantics in the WINGS Workflow System

The WINGS (Workflow INstance Generation and Specialization) workflow system [36, 55, 56, 63, 64] is a workflow composition extension for execution systems such as Pegasus (see [36] and subsection 2.2.4). It features semantic constraints for workflow components. Its origins are from the field of designing text data mining experiments but it has since been used for other applications (see [64]). Figure 8 shows the main user interface with a visual representation of an example workflow and semantic constraints for variables.

The initial stage of a workflow in WINGS is called a template, containing only abstract components and data sources. This is followed by an instantiation stage (where components are made explicit) and an execution stage (where components are mapped to available resources; done in an external workflow execution system).

A workflow template and its components are modelled using an ontology based on OWL-DL (Web Ontology Language – Description Logic). Semantic reasoning is performed using Jena [65] (a modular semantic inference system allowing for different reasoning engines).

Although the original system only used the ontology-based approach to model the workflows themselves, later work exploited the semantic nature to add constraints to components in order to aid the composition process (see [56] for example). Constraints are formulated with RDF (Resource Description Framework) syntax. The following example shows a constraint to restrict the input data of a text analysis method to single-labelled datasets (by defining that multiple labels are invalid) [56]:

```
(?c pc:hasInput ?idv)
(?idv pc:hasArgumentId "Feature")
(?idv dcdom:isMultiLabel "true"^^xsd:boolean)
-> (?c ac:isInvalid "true"^^xsd:boolean)
```

More complex constraints can be specified via Jena rules. The core ontology for workflows and constraints cannot be modified from within WINGS. The user interface



exposes “raw” RDF syntax (such as “Document2 rdf:type dccdom HTMLFile” from Figure 8). Although this approach is flexible and powerful, a user requires deep understanding and knowledge of Semantic Web technologies.

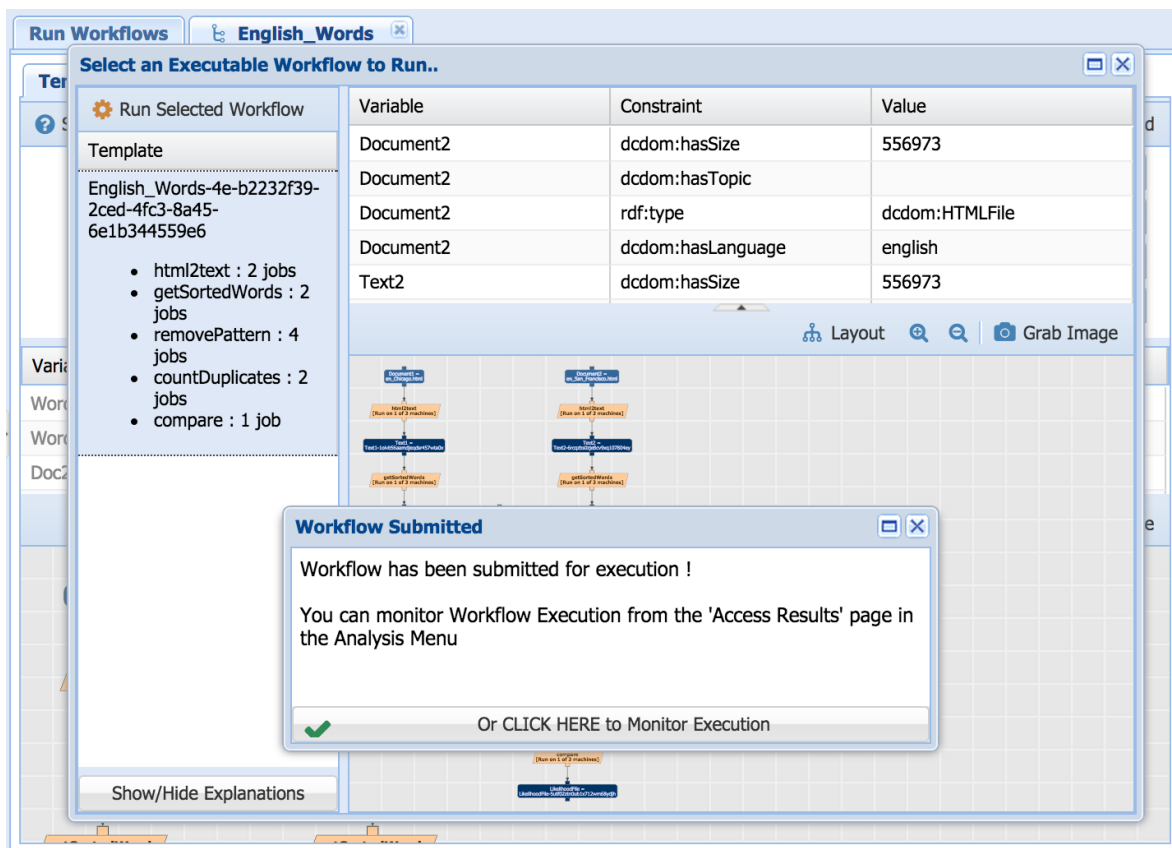


Figure 8 - WINGS workflow composition system ([www.wings-workflows.org](http://www.wings-workflows.org))

## 2.5.2 Semantic Workflow Composition Based on Petri Nets

Petri Nets are a modelling formalism for distributed systems based on directed graphs with nodes for transitions and places (or conditions). Gubala and Bubak [57] describe a Petri-net-based workflow system with semantic features for operation retrieval from a registry (a form of a workflow repository). Every operation is thereby registered via an OWL (Web Ontology Language) description for inputs, outputs, preconditions, and effects.

Gubala and Bubak argue that a registry based on standard taxonomies, such as provided by the UDDI standard (see section 2.4), is “not sufficient to support proper service discovery”

[57]. Their composition system matches different components into a workflow, making use of the semantic data. The tool produces a data flow between components by associating the required input with the produced output. The data matching is separated into three constraints: a content constraint (what is contained), a format constraint (how is it expressed), and a storage constraint (how is it stored). Figure 9 shows an overview of the operation discovery process.

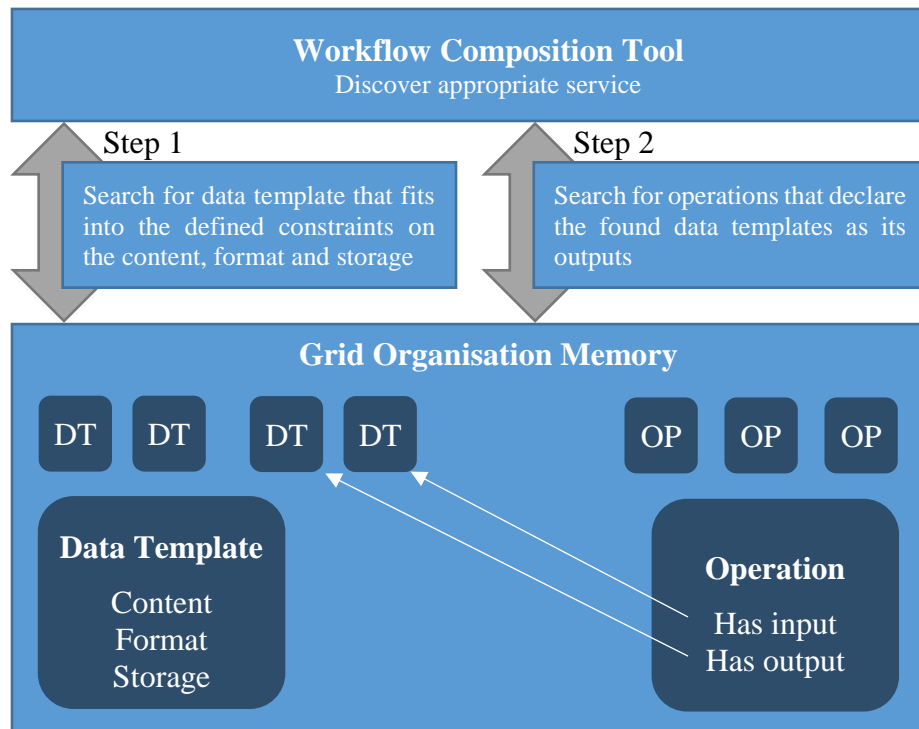


Figure 9 - Operation discovery (adopted from Gubala and Bubak)

The system is Java-based and makes use of open source web services and Semantic Web technologies such as Axis SOAP and the Jena reasoner. Ontologies are managed via Grid Organisational Memory (GOM) [66] and other components. It is designed for purely non-interactive use and therefore does not provide a graphical user interface.

The automated workflow composition uses an iterative algorithm that attempts to fill in unresolved dependencies by using components discovered in a service registry and then

integrating them in a suitable way (as part of a sequence composition, a branching construct (AND / XOR), or within a loop).

The proposed system (a prototype was implemented) uses a strict matching algorithm based on reasoning and therefore requires complete and well-annotated data. Similar to the WINGS system, ontology editing and semantic annotation require expert users, although the final application is targeted at non-expert users.

### 2.5.3 Semantic Features of the ASKALON Workflow System

Within ASKALON (see 2.2.5) [1, 59] semantic concepts are realised as *Activity Functions*, representing abstract activities with semantic metadata. Input and output data of activity functions are *data classes* which can be assigned exactly one semantic meaning (a label). Users can then design workflows using activity functions instead of activity types (i.e. concrete atomic activities in ASKALON). The system can assist the composition process by providing only semantically compatible components.

Ontologies in ASKALON are pure class relationships (“is a” relation). Both data and functions have their own separate class tree.

Abstract activity functions need to be mapped to available concrete activity types before a workflow can be executed in ASKALON. This can be done using an algorithm that matches the input and output data types (or data classes). Partial matches are allowed if a conversion activity is available.

Concrete activities need to be labelled with a semantic function to be considered as a candidate during the matching process. The creator of an activity type therefore needs to carefully assign the semantic meaning as only one definition is allowed (the activity is created for one specific purpose).

On top of the domain-specific ontology sections ASKALON also defines an upper ontology to model the basic concepts such as function and data (see Figure 10). The class “Data Conversion” is used to explicitly denote an activity function to be a data converter of some kind.

Semantic features are stored in OWL and can be edited with an external editor (for example Protégé [60]). Jena is used for reasoning (class relationships).

The semantic annotation scheme in ASKALON is limited as it purely represents an abstraction of activities. An abstract activity (i.e. an activity function) can be assigned only one specific meaning from a tree-like ontology. The matching mechanism, which assigns concrete activities, is strict and only allows partial matches under specific circumstances. Example ontologies in Qin and Fahringer's work [1] have a very narrow application space. They describe mainly a use case for a specific meteorology project. The respective ontology has low reusability value (see Figure 11).

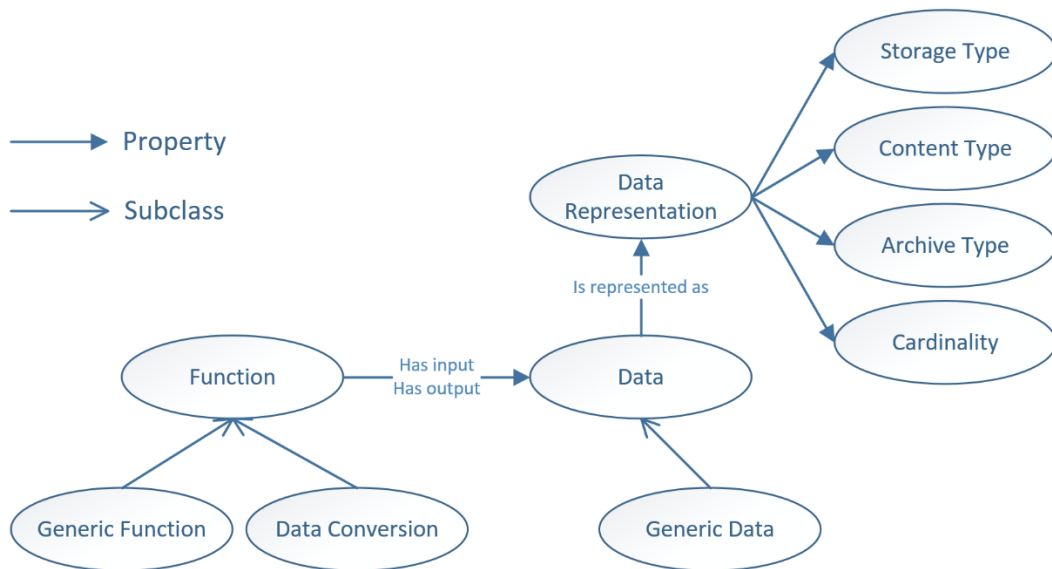


Figure 10 - Upper ontology in ASKALON (see [1])

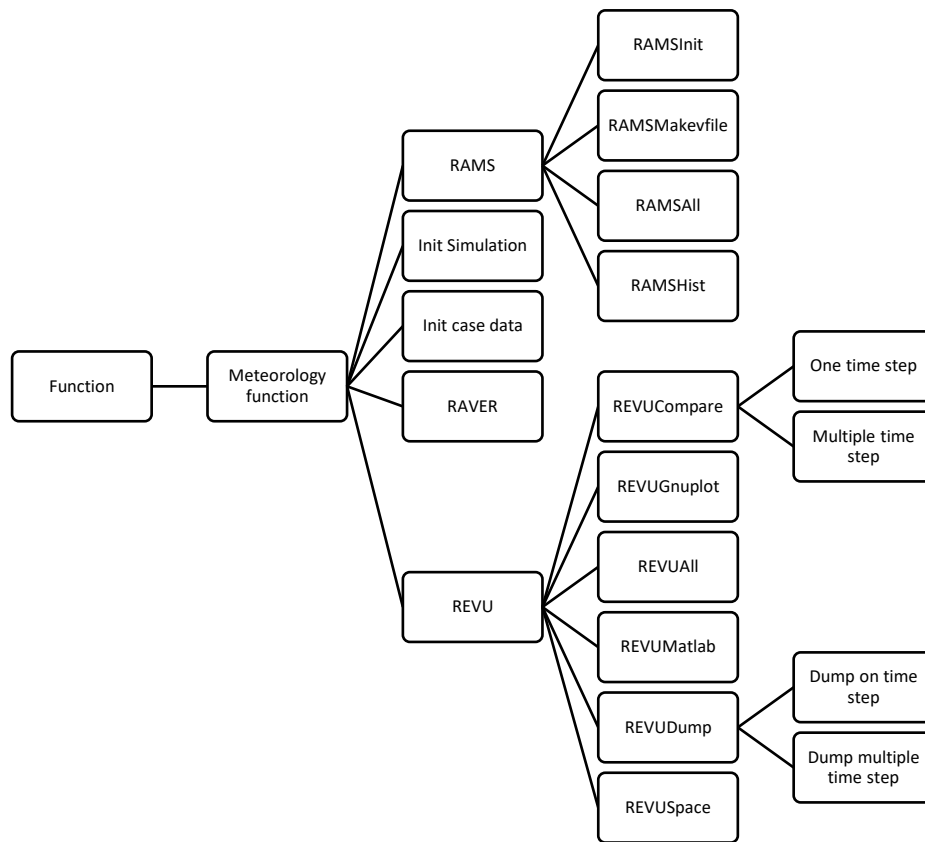


Figure 11 - Example ontology for ASKALON (partial). Meteorology. (after Qin and Fahringer)

#### 2.5.4 Semantic Grid

Wu and Chen [67] developed a theoretical framework called the “Semantic Grid”. They provide an overview of general methodologies from grid computing and the Semantic Web and describe how the two can be combined. Specifically, the following subjects are discussed: knowledge representation (logic, ontology, RDF, OWL, and others), dynamic problem solving (multi-agent systems, ontology management etc.), trust computing, data integration (e.g. semantic mapping and querying), service flow management (matching, composition, and verification), data mining in the semantic grid, and applications.

Wu and Chen present a prototype of a workflow management framework called “Dartflow”. A workflow is therein composed of function units that are defined by a 6-tuple (N, I, O, P, E,  $\psi$ ) where:

1. N is the name of the unit
2. I is the set of inputs
3. O is the set of outputs
4. P is the precondition
5. E is the effect
6.  $\psi$  is the dependency function from the output to the input set

Each functional unit and the whole workflow can be annotated using OWL-S, a semantic mark-up language for web services (see [68]). The annotation is restricted to one semantic class per object (unit, input item, or output item).

The composition process is rule-based and covers three categories:

1. Domain rules: Abstract rules for service selection and composition such as: choice rule, condition rule, exclusion rule, and sequence rule.
2. Business rules: Concrete service binding rules (e.g. preference rule).
3. User rules: Rules for response time and cost of a service.

Wu and Chen argue that it is not feasible to generate a whole business process based on automated service composition using planning methods from the field of Artificial Intelligence (AI). Instead, they propose a composition framework based on a flexible workflow method to create parts of processes using an automated method (see Figure 12). A matching algorithm is used to find suitable functional units based on their inputs and outputs.

Service discovery is implemented using a proprietary request format and reasoning engine.

Wu and Chen describe case studies for traditional Chinese medicine applications and transport management. They focus on the theoretical framework and technical implementation. Practical considerations such as how ontologies are created or how users interact with the system are not discussed. Furthermore, Dartflow seems unavailable for download or purchase and has no apparent dedicated online presence.

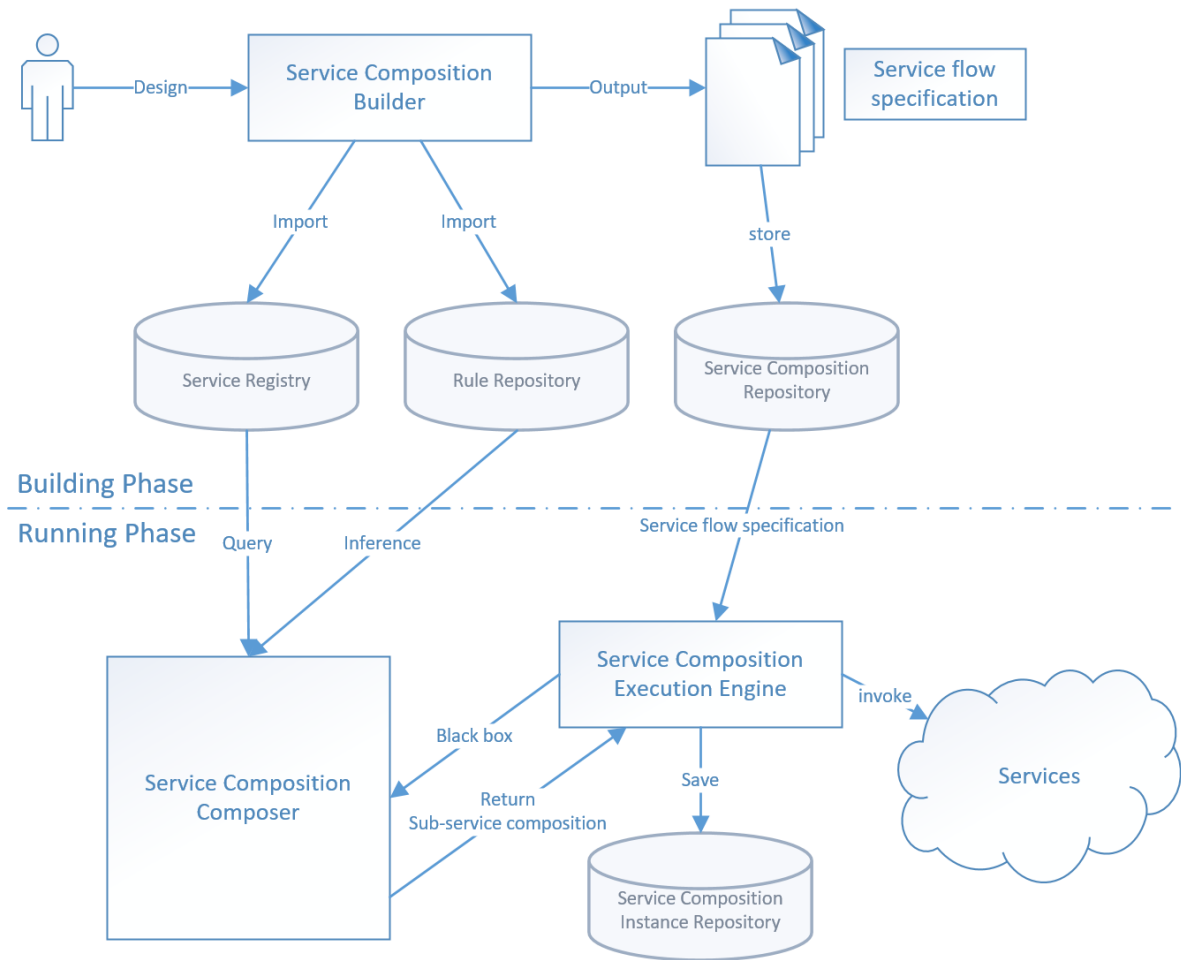


Figure 12 - Service composition framework for the semantic grid (see [67])

## 2.6 Planning Algorithms

To achieve fully automated workflow composition, a method needs to be devised that produces a valid and executable workflow from a set of activities and preconditions. In the field of Artificial Intelligence, planning algorithms are used for such purposes. This section introduces approaches for planning and their possible application to workflow composition.

*State space search* is a general approach where a system or agent is defined by a state and actions to change the state. The goal of the planning is reached when a specific state is reached (see [69]). More specifically, the state space is represented by:

- **S**: a set of all possible states.

- **A**: a set of all possible actions.
- **Action(s)**: a function that returns which actions are available for a certain state.
- **Result(s, a)**: a function that returns the state which results when action  $a$  is performed in state  $s$ .
- **Cost(s, a)**: A function that returns the cost associated with action  $a$  in state  $s$ .

In the context of workflow composition, a state is represented by a workflow, which might be partial. Possible actions are adding workflow components (for instance activities) and connecting data sources. A search is complete when an executable workflow is found that outputs the desired data and (optionally) adheres to predefined constraints.

Given the large amount of possible actions (hundreds or thousands, depending on the number of available atomic activities and their data ports), the search space dramatically increases in size with each action. An exhaustive search is therefore unrealistic for anything but trivial workflows.

Ambite and Kapoor [70] use a partial-order planner (for general information see [71]) to create a data workflow that satisfies a user's request. Their algorithm keeps an *agenda* of services (activities) with unachieved inputs. Each search step refines the plan by satisfying one missing input from the agenda, using a semantic matching service. An empirical analysis (by Ambite and Kapoor) shows that the algorithm is effective, but also indicates that the planning time can be prohibitively long for large search spaces.

A complete automation of workflow composition using planning algorithms seems unfeasible at this time (see [62]). Rather efforts should be focussed on solving sub-problems such as choosing suitable workflow templates from a repository, adding data conversion activities (shims), or creating processing sequences (node-to-node processing without branching).

## 2.7 System Design and Advanced Features

When designing and developing a workflow system and/or repository, certain guidelines should be followed. The most prevalent design strategies are described in this subsection.



Surveying the literature, it becomes apparent that there is a plethora of models and languages for scientific workflows. It is therefore sensible not to create something completely new but to keep to standards or widely accepted concepts. This view is shared by Barker and van Hemert [16]. They also propose further research directions including, but not limited to: collaboration between multiple domains, consideration of conventional scripting languages (e.g. Perl), high level of abstraction, and favouring web-based solutions.

Different user groups may require different views on a workflow system. Ludäscher et al. [19] propose three levels of abstraction: (1) Conceptual level (workflows might become executable only after refinement); (2) Analytical level (knowledge discovery workflows for scientists); (3) Low-level workflows (for grid engineers).

In document image analysis the above view hierarchy could be adapted to high-level view (e.g. for librarians), mid-level view (e.g. for researcher), and low-level view (e.g. for IT expert). A fourth conceivable user group are workflow system experts that can make adjustments to or extend the workflow model. Nadarajan et al. use the term “Modeller” [58].

*Workflow templates* represent one way of a conceptual-level of abstraction. Templates are blueprints and cannot be executed until concrete workflows have been created from them. The WINGS system is entirely based on templates (see Section 2.5.1) and the ASKALON system allows the creation of templates in form of *activity functions* (see Section 2.5.3).

Advanced features of a workflow system can include *workflow optimisation* and *simulation*. Curcin et al. [72], for example, use a stochastic model to perform execution simulations of Taverna workflows. Jansen-Vullers and Netjes [73] surveyed several business process simulation tools based on Petri nets and event-driven process chains. Chen and Deelman [74] propose to simulate workflows to estimate runtime performance, including system overheads and (random) failures.

## 2.8 Workflows in Document Image Analysis

Only relatively recently scientific workflows have started to be of significant interest within the document image analysis research community. Two different approaches using the Taverna system were reported in 2011. Lamiroy and Lopresti [75] applied workflows to

achieve an open end-to-end architecture for benchmarking. The other approach, by Neudecker et al. [27], was developed as part of the EU-funded IMPACT project [76]. Several tools developed within the project were combined into the Interoperability Framework. Taverna workflows allow for setting up different experiments and production pipelines. Figure 13 shows a workflow that has been created using the Taverna Workbench tool. As mentioned earlier, the workflow designer needs to be aware of available processing nodes and their function as well as data compatibility (input and output formats).

In order to be able to create a model that is beneficial for the composition of workflows in document image analysis, all aspects of the field have to be taken into account in detail. Only if the model is as complete as possible, it is likely to find acceptance among domain experts (and other users). The next chapter provides an overview of semantics and presents a new model for document image analysis.

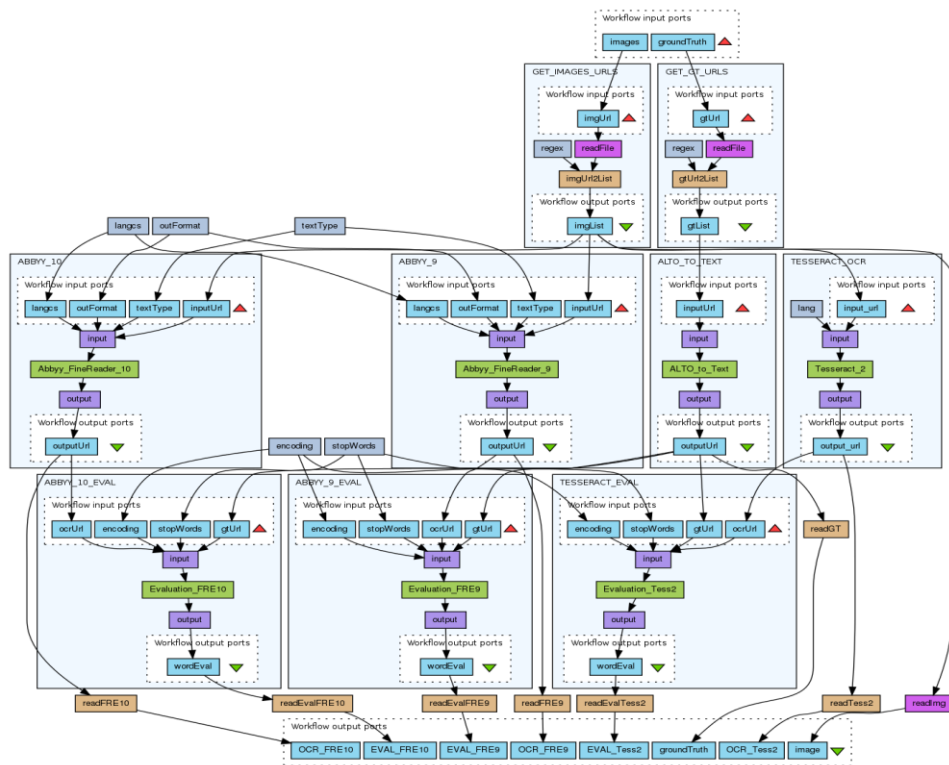


Figure 13 - Taverna workflow for comparing Optical Character Recognition (OCR) engines ([27])

## 2.9 Summary

This chapter introduced workflows and their use to model and execute experiments or processes in general. Different modelling approaches, workflow systems, and workflow repositories were discussed (Taverna, Kepler, Triana, Pegasus, ASKALON, myExperiment etc.).

It was pointed out that system complexity and lack of automation are a hinderance for users. Integrating semantic information and using it for workflow composition and management was presented as possible solution. Several concepts and systems with semantic features were outlined (e.g. WINGS).

The chapter concluded with a discussion of planning algorithms for workflow composition, concepts for workflow system design, advanced system features, and recent work in document image analysis with respect to workflows.

The next chapter focusses on engineering an ontology for document image analysis for use in workflow systems.

## 3 A Semantic Model for Document Image Analysis and Recognition

Knowledge of a domain must be incorporated into a formal model to be usable for workflow composition or related tasks. This chapter first provides an overview of semantics and knowledge representation in general and then describes the development of a new model for the domain of document image analysis.

### 3.1 Knowledge Representation and Semantics

The overarching goal of creating models and encoding information is the representation of a part of the real world in order to better understand it or process it in some form. Although scientific workflow languages are modelling approaches, they mainly focus on the *syntactic* aspect of combining actors (activities) in a certain way to automate the execution of a process such as a scientific experiment. To enable a machine to assemble such workflows, the *meaning* of each available component has to be described in machine-readable form.

Various definitions of the term *semantics* exist in the context of different fields of application (for instance linguistics and scientific classification). For this work, the concepts from artificial intelligence and the Semantic Web are of interest, wherein semantics is the explicitly expressed intended meaning of a resource so that it can be processed by a computer (or machine). Central to this is the term *ontology*, which can be defined as “a description of knowledge about a domain of interest, the core of which is a machine-processable specification with a formally defined meaning.” [77]

Many ideas from the Semantic Web can be directly applied to scientific workflows. Especially the Web Ontology Language (OWL) and the modelling paradigms it entails are of interest in this work. Hitzler, Krötzsch, and Rudolph [77] provide an extensive theoretical and practical discussion of OWL including the Resource Description Framework (RDF) it is based on. In accordance with the concepts mentioned above, they list three keystones of the Semantic Web:

- “*Building models*: the quest for describing the world in abstract terms to allow for an easier understanding of a complex reality”.
- “*Computing with knowledge*: the endeavour of constructing reasoning machines that can draw meaningful conclusions from encoded knowledge”.
- “*Exchanging information*: the transmission of complex information resources among computers that allows us to distribute, interlink, and reconcile knowledge on a global scale”.

While other approaches to represent information exist, the solutions of Semantic Web are ideal for the application in workflow systems because both share the ideas of global resources and applications of artificial intelligence. Furthermore, several standards have been defined for representing semantic information, enabling the use of existing modelling tools and reasoners (a reasoner infers consequences from factual knowledge).

The next two sections provide an introduction to ontologies and an overview of document image analysis and recognition (the target domain), followed by the description of the methodology and the creation of a semantic model that can be used for workflows.

## 3.2 Ontology Engineering

For a comparatively long time, the creation of ontologies followed only informal guidelines. There was no specific methodology for the design or evaluation of an ontology. This has changed over the recent years and now the term *ontology engineering* is preferred ([78, 79]). In this section, it is outlined how to approach a new ontology in a methodical way.

An ontology can be referred to as “the shared understanding of some domain of interest which may be used as a unifying framework” [78], enabling:

- Improved communication.
- Inter-operability.
- Re-use and sharing.

An ontology always includes a vocabulary of terms and a specification of their meaning (definition).

Uschold and Gruninger [78] outline guidelines for building ontologies, taking into account: identification of purpose and scope (scoping phase), ontology capture, ontology coding, integration of existing ontologies, evaluation, and documentation.

The *capture* is therein described by three steps:

1. Identification of key concepts and relations in the target domain.
2. Defining the concepts and relations in an unambiguous way.
3. Identifying terms that represent the concepts and relations.

*Coding* is the formalisation of the outcome of the capture phase, using a specific ontology language. This includes also the commitment to basic terms such as “class” or “relation”, sometimes called a *meta-ontology*.

In addition, they provide general guidelines, stressing principles of:

- Clarity (provide distinctions where ambiguity can occur; provide examples; use natural language for definitions).
- Coherence (i.e. internal consistency).
- Extensibility (selection of vocabulary with sharing and future extension in mind).

The *scoping* phase can be organised as a brain-storming session, where all related terms are collected, followed by a grouping stage where the terms are categorised for inclusion/exclusion and grouped by similarity.

When producing the definitions of terms, it is recommended to favour a middle-out approach over a top-down or bottom-up strategy. That means the most fundamental terms in each work area are used as a starting point from which is then moved forward towards more specific as well as toward more general terms.

The next subsection introduces four ontology engineering strategies. The one called METHONTOLOGY[3] is endorsed by Fernández-López [79] as the most mature of the reviewed approaches.

### 3.2.1 Ontology Creation Methodologies

Fernández-López [79] reviewed several methodologies for building ontologies. He relates the IEEE Standard 1074 for software engineering to ontologies on the basis that they are (or are part of) software products. He uses nine criteria for the analysis, including but not limited to: detail of specification, strategies for building ontologies, life cycle, and ontologies developed with the approach. Four methodologies are outlined below (most detail given for METHONTOLOGY).

Uschold [80] proposed a design approach based on the experience of creating an ontology for enterprise modelling processes. Design guidelines are provided for: identifying the purpose, building (capture, coding, and integration with other ontologies), evaluation, and documentation. The methodology is incomplete with regards to life cycle and offers little detail. [79]

Grüninger and Fox [81] also formalised their approach based on experience in creating a business-related ontology. The process is to move from informal to formal using the following steps: capture of motivation, formulation of informal competency questions, specification of terminology, rephrasing the competency question using the terminology, specification of axioms and definitions, and characterisation of completeness.

In common with Uschold's approach, this methodology lacks detail and the definition of a life cycle. [79]

The SENSUS-based methodology [82] makes use of a large knowledge base for machine translation (the SENSUS ontology [83]). The proposed steps are: specify seed terms, link terms to SENSUS, include all terms from seed to SENSUS root, add missing terms, and expand to full subtrees (where many seed terms appear in a subtree).

The SENSUS-based approach also omits the definition of a life cycle [79]. In general, it is the most unique of the compared methodologies, but it is inherently limited by the domain coverage of the SENSUS ontology.

The METHONTOLOGY framework [3] includes techniques for ontology development and a life cycle description. Fernández-López et al. [3] draw from their experience in knowledge engineering (e.g. expert systems) but note that there is a main difference in comparison to ontologies: Knowledge-bases can usually be built up incrementally in many cycles. Ontologies, on the other hand, need to be more complete from the beginning because they are built to be shared and reused widely.

The development process for ontologies is defined by the following steps:

- Planning (main tasks to be done and their order; time; resources such as people and software).
- Purpose and scope definition (“Why is this ontology being built?”, “What are its intended uses and end users?”, the answer should be part of a requirement specification document).
- Knowledge gathering (including listing of the sources used).
- Conceptualisation (building a conceptual model from the gathered knowledge, describing problem and solution).
- Formalisation (transformation of the conceptual model to a formal one).
- Integration of existing ontologies (to avoid duplication and assist in reuse).
- Implementation (make the ontology machine readable via a formal language).
- Evaluation.
- Maintenance (e.g. extension or modification).

Just as software projects, ontologies should be engineered using a proper life cycle strategy. Figure 14 shows the METHONTOLOGY life cycle with the aforementioned activities and stages. Other approaches, such as the waterfall life cycle, are considered inadequate for ontologies because of their evolving nature. While waterfall and basic incremental life cycles cause problems, the proposed method of *evolving prototypes* can cope with growing ontologies (see Figure 15). This approach allows the ontology to grow depending on the needs: “This model lets you modify, add, and remove definitions in the ontology at any time.”[3]



The formalisation step of METHONTOLOGY does not specify a formal language that is to be used. Formal languages are discussed in the next subsection.

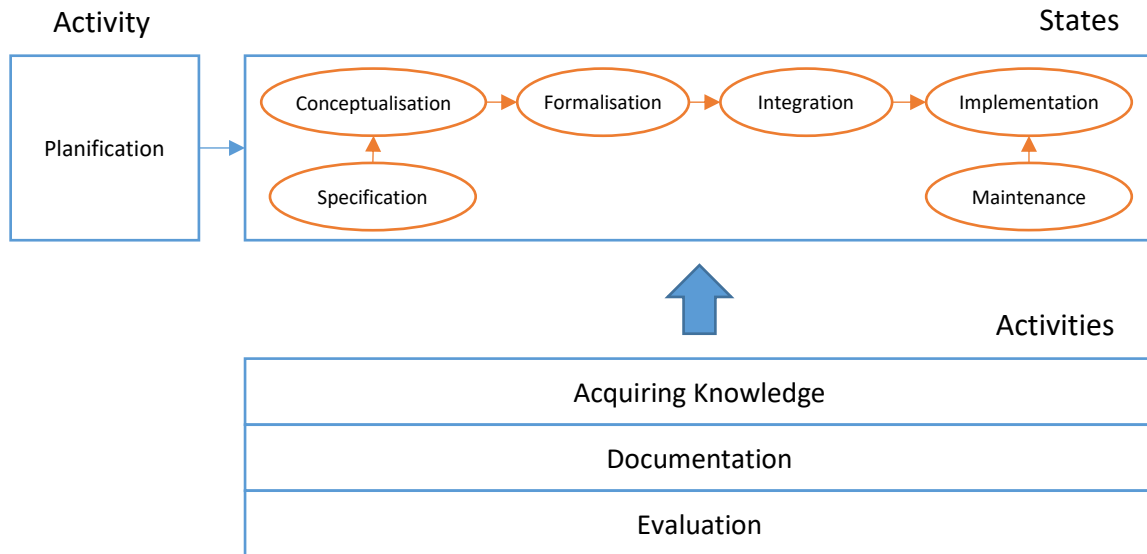


Figure 14 - States and activities of METHONTOLOGY (adopted from [3])

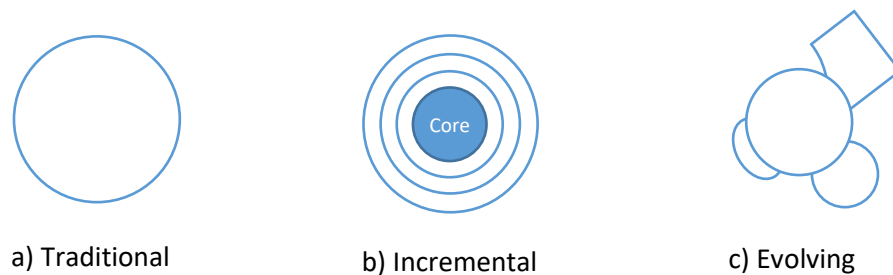


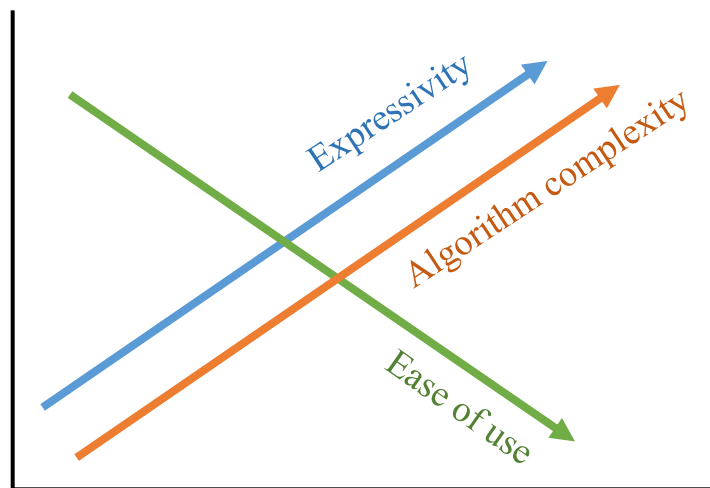
Figure 15 - How the ontology grows: a) Fixed / waterfall; b) Basic incremental; c) Evolving prototypes (adopted from [3])

### 3.2.2 Meta-Ontology, Formal Language and Expressivity

To make an ontology machine-readable, it has to be translated to a formal language. As mentioned earlier, part of this step is also the selection of a suitable *meta-ontology*.

For an informed decision on the basic structure of the ontology, three aspects must be considered: *expressivity*, reasoning algorithm *complexity*, and *ease of use*. The more

expressive the ontology, the closer it can reflect the real world and model detailed relationships and interdependencies. However, with increased expressivity there is also an increased algorithm complexity for reasoners (inference engines). This can be a problem for systems that offer real-time user interaction, such as workflow composition systems. In addition, the importance of the ease of use should not be underestimated. A system that is intrinsically complex and difficult to understand is less likely to be adopted by users in comparison to a system that uses a straightforward semantic model. Figure 16 illustrates these considerations.



*Figure 16 - Expressivity, algorithm complexity, and ease of use of ontologies and systems using them*

Obrst et al. [84] describe three ontology representation levels (Figure 17) which provide an overview and help to find a distinction between the terms being used.

To implement an ontology, a suitable language must be chosen. Traditional options are CLASSIC, BACK, LOOM, or Ontolingua (see [3]). Alternatively, an ontology can be coded with any higher-level programming language (such as C++ or Java). In the context of the Semantic Web the Resource Description Framework (RDF) or Web Ontology Language are used (see [77]).

METHONTOLOGY includes evaluation as a distinct action but does not provide a specific evaluation method. Therefore, an overview of ontology evaluation approaches is given in the next subsection.

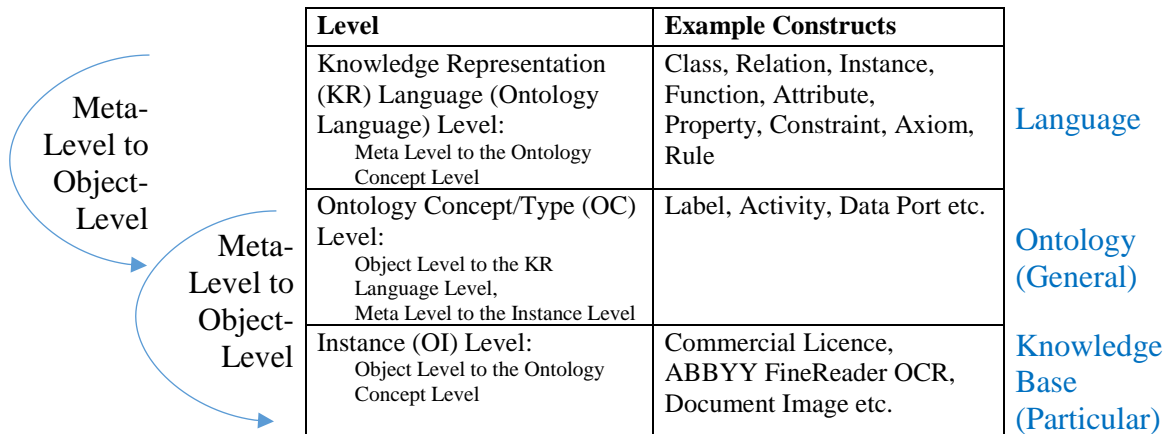


Figure 17 - Ontology representation levels (adapted from Obrst et al. [84])

### 3.2.3 Evaluation of Ontologies

Obrst et al. [84] describe several approaches or perspectives on how to evaluate an ontology, including:

- Domain coverage.
- Complexity / granularity.
- Use cases / scenarios / applications / data sources.
- Consistency.
- Completeness.
- Representation language (e.g. expressivity).

In addition, they make the distinction between a component-based “glass box” approach and a “black box” approach. The black box approach can be especially useful if the ontology is to be evaluated in combination with a specific software system (like a semantic search engine).

One evaluation criterion is of special relevance for the research described in this Thesis: use cases and domain requirements. The validation of the workflow system prototype using real-world examples is one of the objectives of the PhD research. It is therefore sensible (even unavoidable) to include the developed ontology as well in this examination.

Obrst et al. [84] write that the same competency questions that are used during requirement specification and conceptualisation can also be used to validate the finished system (representing a form of a “test suite”).

### 3.2.4 Ontology Maintenance

In most applications, even if an ontology is complete, it requires constant attention in the form of correcting flaws, refining descriptions, and extending concepts. In METHONTOLOGY this is part of the life cycle and is called *maintenance*. A dedicated *maintainer* therein includes, adds, or modifies existing definitions as required. The life cycle model of evolving prototypes supports these activities.

Flahive, Taniar, Rahayu, and Aduhan [85] describe a process of ontology update tailored for grid workflow environments (i.e. distributed workflow execution). They formalise and validate the replacement of a section of an ontology  $O_2$  with a subset of another ontology  $O_1$  using two phases:

1. Extraction of subset  $S_1$  from  $O_1$ : Creating a subontology, refining it using four operations: Extend, add, merge, and update.
2. Replace concepts in  $O_2$  by  $S_1$ : Discovering merge points and replacing parts of  $O_2$  with the subontology, finalised using validity checks.

Before describing how an ontology was created, a short summary of the target domain is given in the next section.

## 3.3 Document Image Analysis and Recognition

An overview of the domain helps to understand the key concepts used in the ontology creation. This section provides basic definitions, examples, and use cases.

### 3.3.1 Overview

O’Gorman and Kasturi [86] and Ferilli [7] provide comprehensive overviews of the field, including data formats, image processing, segmentation, document understanding, and natural language processing. O’Gorman and Kasturi state the objective of *document image analysis* as “to recognize the text and graphics components in images, and to extract the intended information as a human would”. Ferilli provides a more abstract definition for document image analysis to be “concerned with the automatic interpretation of images of documents”[7].

Figure 18 shows a typical scenario for document image analysis. Often the term *digitisation* is used synonymously for a whole image analysis pipeline. The original meaning, however, is the conversion from a physical document to digital form via *image acquisition* (scanning, digital photography). Image acquisition also covers the conversion of other digital formats (such as PDF or HTML) to image formats as is required for *digital-born* documents.

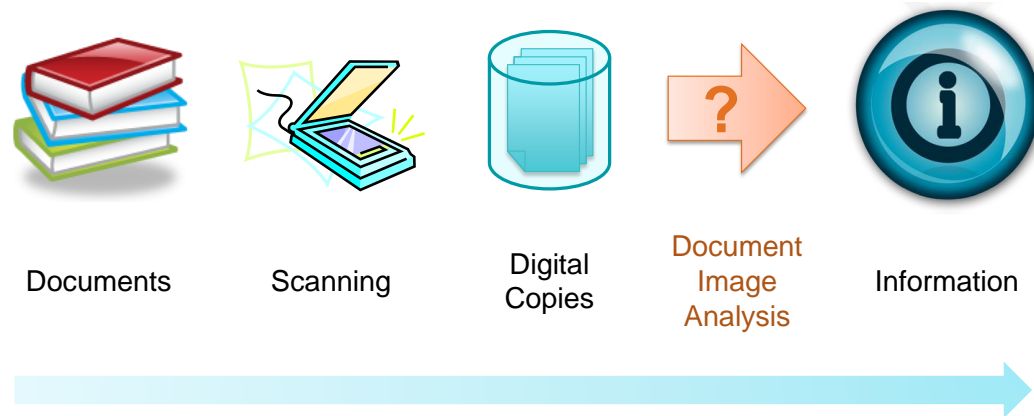
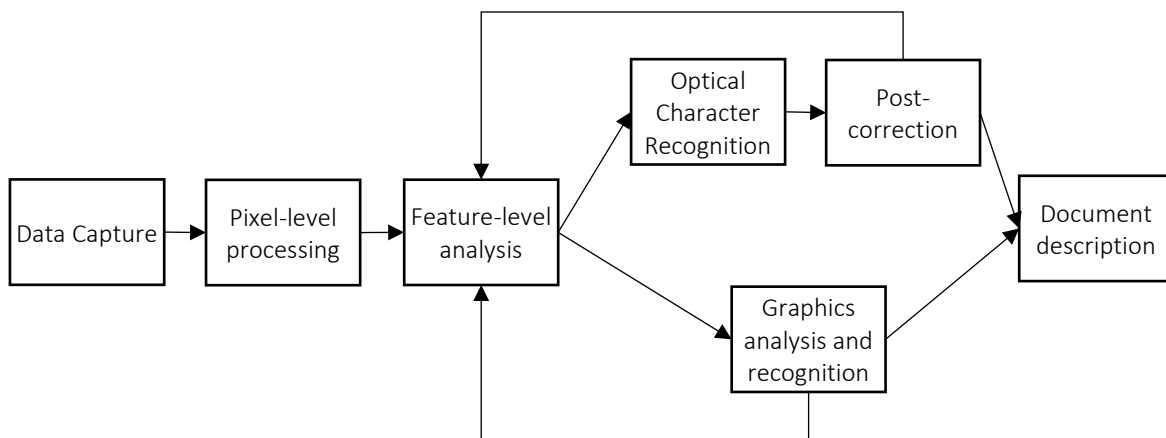


Figure 18 - Digitisation scenario

O’Gorman and Kasturi as well as Ferilli offer extensive definitions of the term *document* and *document image*. For this work, it should suffice to provide the following examples that can be represented by digital images (contrary to speech, for instance): Book, newspaper, Greek roll, picture / photograph, webpage, bank cheque, number plate, and shopping receipt.

Ferilli organises documents by the following categories:

- Support – Tangible / physical (written, printed, impressed etc.) and intangible (digital documents, speech, music etc.)
- Time of production
- Historical interest
- Medium
- Structure – The meaningful arrangement of document components to form the document as a whole and/or the inner structure of a single component
- Representation formalism – Related to the target audience/interpreter (human or computer)



*Figure 19 - Typical sequence of steps for document image analysis (adapted from [86])*

Figure 19 shows an often-used sequence of steps as described by O’Gorman and Kasturi [86]. One of the best-known fields within document image analysis is Optical Character Recognition (OCR). Other major areas are: page layout analysis, segmentation, graphics recognition, and document restoration. There is also a considerable overlap with other disciplines such as the related document analysis, linguistics, image processing, pattern recognition, and machine learning.

Although often called digitisation “pipelines” (indicating sequential processing), typical workflows include branching (specialised processing for different types of documents) and feedback loops (for improving earlier steps based on extracted information).

A few often-used processing steps are explained below.

### 3.3.2 Typical Processing Steps

In this subsection selected processing steps are presented because they apply to many different types of workflows (e.g. image pre-processing) or because they apply to a very prevalent class of documents (e.g. OCR used for textual documents).

#### **Image pre-processing / pixel-level processing**

Pre-processing is applied to enable or enhance subsequent processing steps. It is a low-level form of processing, considering little or no information of the image content. Both input and output are digital images. Typical methods include: binarisation (conversion from colour or greyscale to bitonal black-and-white), cropping, rotating, and contrast enhancement.

#### **Page layout analysis / segmentation**

Segmentation is the process of subdividing an image (area) into smaller homogeneous areas. The criteria for homogeneity vary for different applications but usually include texture, foreground density, or colour / grey value properties.

A document page can be segmented into *blocks* (regions, zones). In combination with block classification we speak of *page layout analysis*. This can also include higher-level structures like nesting of blocks, layers, and reading order (sequence of blocks).

Blocks can be segmented further into text line segments and word segments, for example.

#### **Optical Character Recognition (OCR)**

Also known as text recognition, OCR is the process of recognising the shapes of characters and assigning a code (or class) according to a text encoding scheme (e.g. Unicode). OCR methods often include segmentation to some extent (text lines, words, shapes / glyphs).

Output of an OCR method can be in plain text format (just characters and whitespaces) or in a structured format with metadata and annotations.

### 3.3.3 Data Formats

Data is central to scientific workflows. Data formats are therefore of special interest for this PhD research. Common input, intermediate, and output formats include:

- Image files with:
  - Different encodings.
  - Different compression rates (none vs. lossless vs lossy)
  - Different colour ranges (bitonal, greyscale, colour)
- Text files with:
  - Different encodings (e.g. Unicode)
  - Plain text vs. annotated text (i.e. with tags for different types of annotation)
- XML-based files like:
  - ALTO XML [87] (for OCR results including layout elements and recognised text with metadata)
  - PAGE XML [88] (for ground truth of page content including layout elements with attributes, transcribed text, reading order and more)

### 3.3.4 Applications

Important real-world examples can provide use cases for scientific workflows in document analysis and give direction for future research. The Google Book Search project [89] falls in this category. A complex pipeline was developed, including components for page ordering, language identification, chapter detection, book clustering, and more. Another example for a large-scale project is “Digitizing a Million Books” by Sankar et al. [90]. Apart from a description of the manual and automated processes that were applied, the authors provide a list of open research challenges, including: language/script-related issues, OCR quality, robust search, compression and delivery, historic media (e.g. palm leaves), and non-textual content.



Digitisation pipelines and their evaluation can lead to complex workflows. Figure 20 shows the evaluation workflow used by Pletschacher et al. [91] in the Europeana Newspapers Project. In context of performance evaluation *ground truth* is the perfect result of an automated processing method. Ground truth must, by definition, be created (or at least validated) by a human. Performance evaluation is crucial for comparing methods but also for improving methods (development or training).

If semantic information is to be used to help with workflow creation, a common terminology must be developed and organised. This is discussed in the next section.

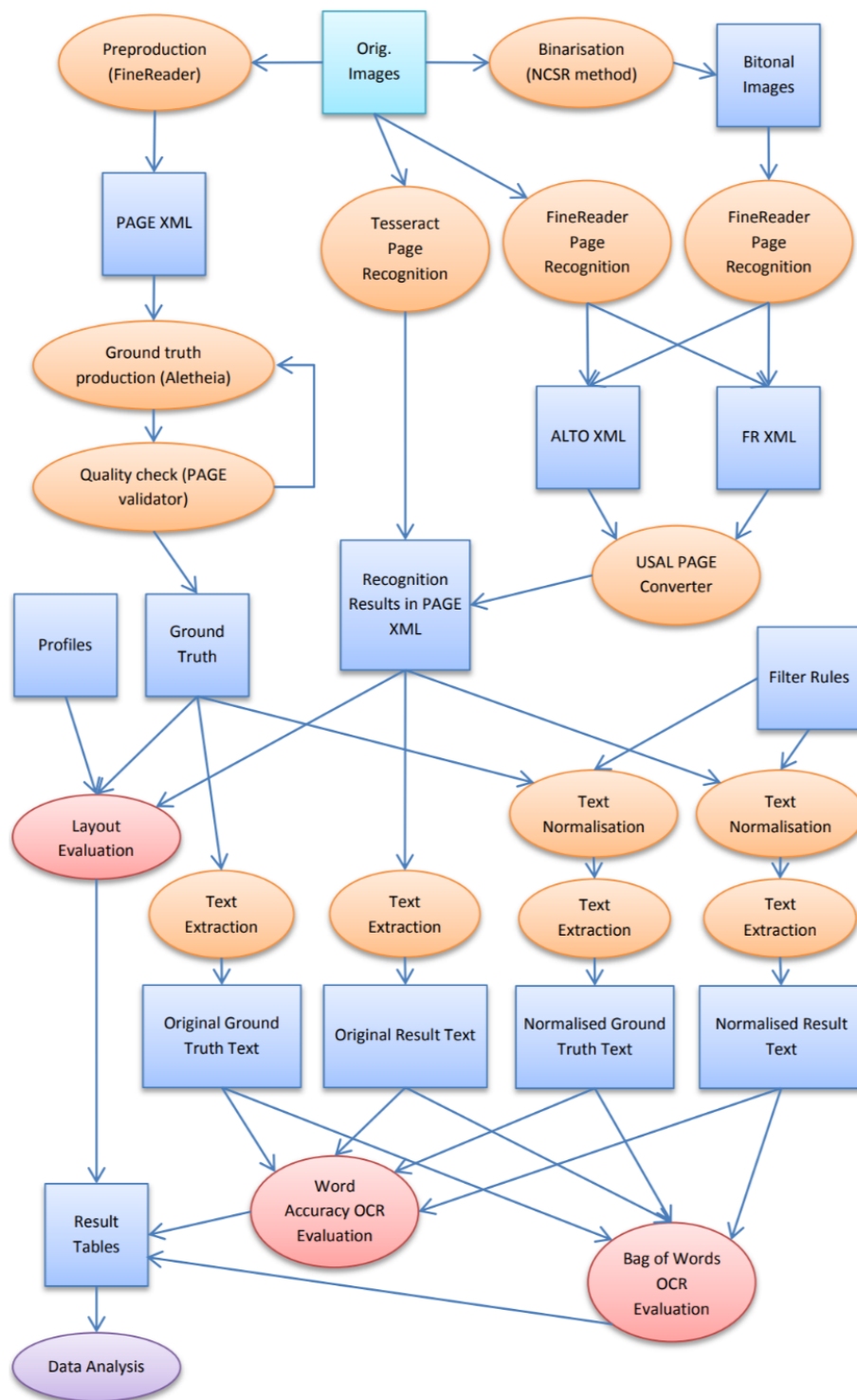


Figure 20 - Evaluation workflow of Europeana Newspapers Project [91]

## 3.4 Ontology for Document Image Analysis

This section describes how a new ontology for the domain of document image analysis and recognition was created. It follows the ontology engineering methodology of choice (METHONTOLOGY [3]) from Section 3.2, chosen for its maturity, completeness of life cycle approach, and general applicability (e.g. to different domains such as chemistry and linguistics) [79].

### 3.4.1 Specification

This subsection contains the ontology specification document as suggested by METHONTOLOGY.

**Domain:** Document image analysis

**Date:** June 28<sup>th</sup>, 2018

**Purpose:** Ontology to describe workflows in document image analysis, including scientific experiments and production digitisation pipelines.

**Level of formality:** Semi-informal

**Scope:** Methods and data objects related to document image analysis, including overlapping domains such as document analysis, image processing, pattern recognition, natural language processing, machine learning etc. The granularity has to be high enough in order to enable automated reasoners to decide which components to use to create a workflow for a specific goal (experiment, pipeline). Target users will be domain experts (researchers), method creators, and non-experts (regarding the domain; e.g. librarians).

### 3.4.2 Knowledge Acquisition

This step is not strictly a task within the ontology development life cycle, but it is important nonetheless. It is part of the documentation process, leading to a comprehensible outcome.

Information about scientific workflows was ascertained via the literature review for this PhD research. The focus for additional knowledge acquisition was to identify sources of classification and structures for the domain of document image analysis and related fields. The following resources were considered:

**Existing ontologies:**

- The ACM Computing Classification System [92].

**Text books:**

- “Automatic Digital Document Processing and Management” [7].
- “Digital image processing” [93].

**Project outcomes:**

- “Digitisation Tools Matrix” created by the SUCCEED project [94].
- Keyword collection and categorisation used for tagging document image datasets in the EU-funded projects IMPACT [95] and Europeana Newspapers [96].

**Web resources:**

- Wikipedia<sup>3</sup> (domain-related terms and definitions).

**Proceedings:**

- International Journal on Document Analysis and Recognition (IJ DAR) (Collection of terms taken from journal paper titles from 1998 to 2016 (19 volumes)).

### 3.4.3 Conceptualisation

This step is used to build up a complete vocabulary for the domain, mainly by collecting and grouping terms (concepts, instances, verbs, and attributes).

The first action was to arrange terms from the resources that were gathered in the knowledge acquisition step into a “term cloud” (Figure 21). These terms were collected from paper titles, table of contents, definitions, taxonomies, and similar sources. This provides a

---

<sup>3</sup> <https://www.wikipedia.org>



For the grouping phase, it was first considered where the information of the ontology will be applied within a workflow system. It was known from the literature that all workflow representation paradigms have in common that they are structured as some form of a graph containing *activity* elements (actors, processes) and *data* objects (or data connections, ports, cables).

Since the intention is not to replicate the workflow structure itself in the ontology, a labelling approach was considered a good starting point. Therein semantic labels would be assigned to workflow components, to enrich them with ‘meaning’. Consequently, the term cloud was further refined in two branches: one concentrating on workflow activities and the other one on workflow data objects.

Each of the two branches of the term cloud was refined iteratively by:

- Removing irrelevant terms.
- Spatially grouping related terms.
- Spatially arranging groups (related groups close together).

Then, terms were inserted or highlighted that represent a group or a sub-group. Figure 22 shows the term cloud after refinement for workflow activities. Table 2 lists the terms that represent the main groups found by the conceptualisation process. Figure 23 provides more detail (sub-groups) of the activity related terms.

*Table 2 - Root ontology terms for workflow activities and data objects*

<b>Activity related terms</b>	<b>Data related terms</b>
Activity domain	Source
Processing level	Age
Data Creation / Transformation	Physical production method
Adaptability / Applicability	Acquisition method / replication steps
Automation	Precision
Licence	Content type
Platform	Content encoding
Maturity	Source / target content
	Data granularity
	Data condition
	Data attributes
	Topic

The formalisation step (creating a formal or semi-compatible model using a representation system) from METHONTOLOGY was merged with the implementation step (creating a machine-readable model). Software tools can be used to visualise and edit ontologies on-the-fly, making a dedicated manual formalisation unnecessary.

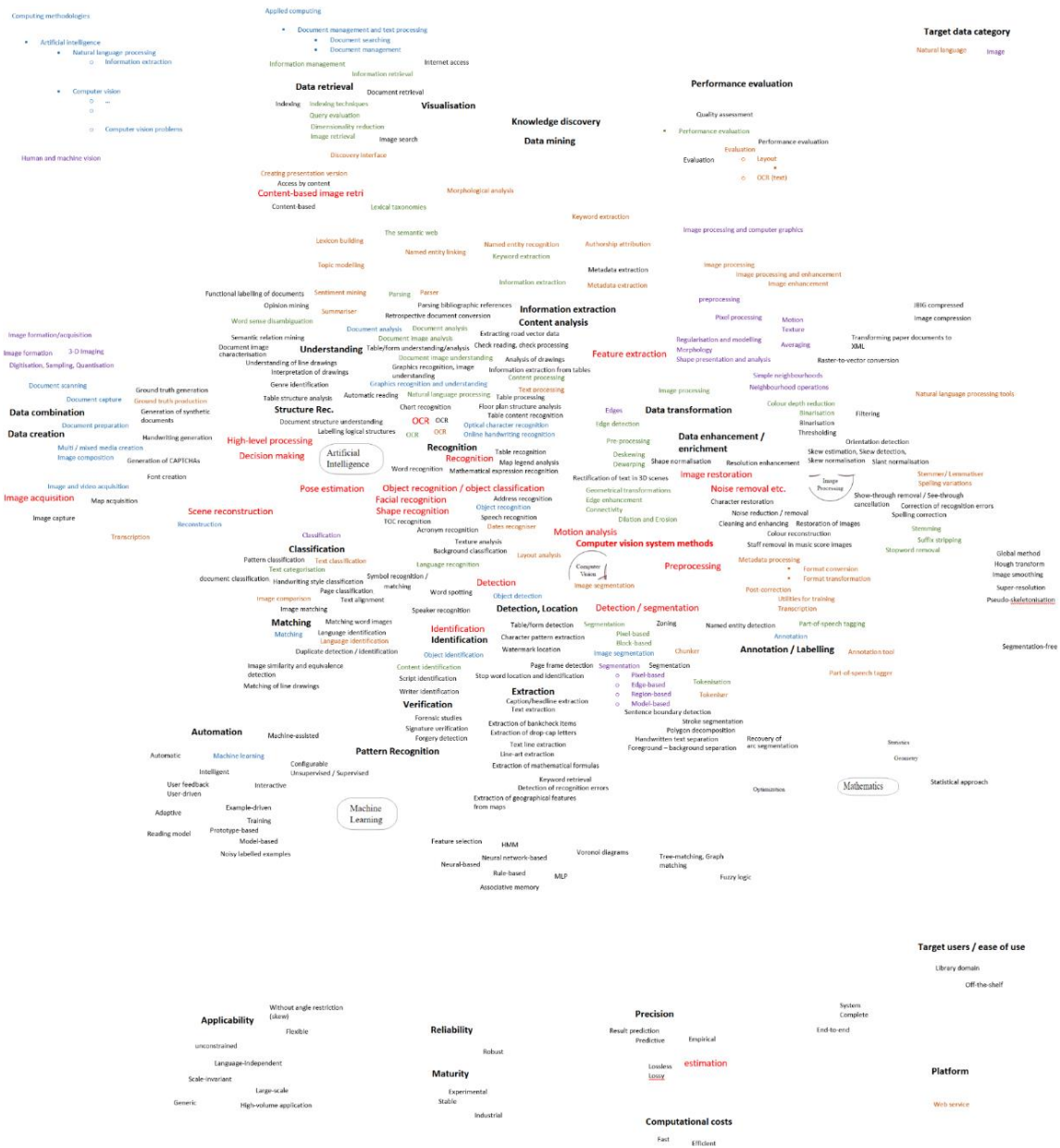
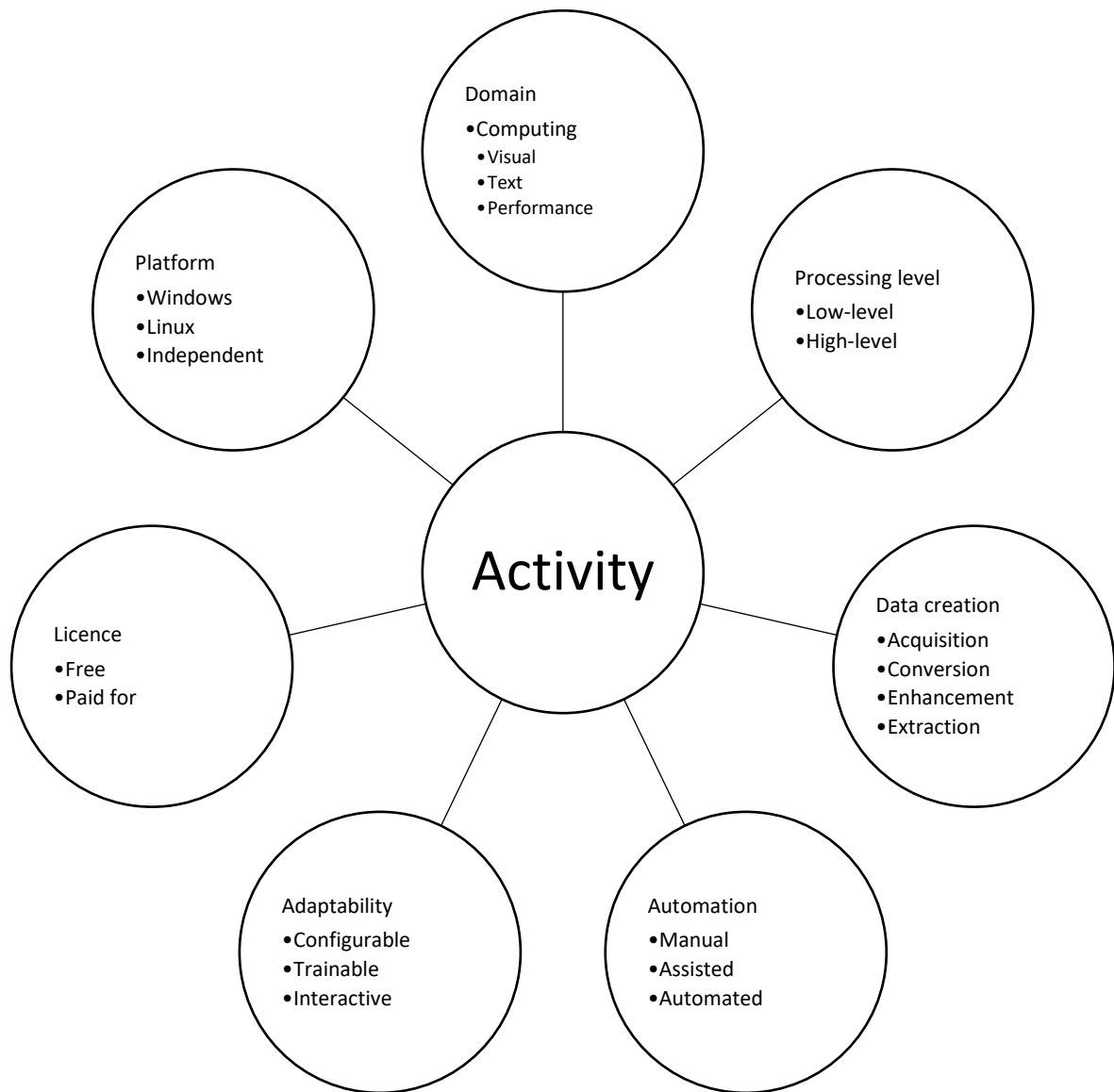


Figure 22 - Refined term cloud for workflow activities



*Figure 23 – Activity-related ontology terms*

### 3.4.4 Integration

As stated in the knowledge acquisition step, the ACM Computing Classification System [92] was considered as input for the new ontology. The system is a poly-hierarchical ontology including base concepts such as: hardware, computer systems organisation, networks, software and its engineering, theory of computation, mathematics of computation, information systems, security and privacy, human-centred computing, computing



methodologies, applied computing, and social and professional topics. It covers a wide range of computing-related terms and was therefore an ideal source for reusing concepts. One example is the branch “Computing - Information systems – Information retrieval” which was adopted as “Activity domain – Computing – Information management – Data retrieval”.

### 3.4.5 Implementation

It was already decided earlier to use a labelling approach (workflow components labelled using the ontology), but it was not decided how complex the ontology should be (i.e. what basic types of concepts and relations are to be used). The strategy adopted was therefore as follows: create an elementary ontology first and test if it is expressive enough for the intended use for automation in workflow creation (Chapter 6). It could then be extended as required.

The most basic approach would be to only use the identified terms without adding any structure, essentially creating a keyword-based solution. Because this would also complicate the use of the system (users would have to browse hundreds of possible keywords), a hierarchical structure was chosen. Since the goal is the labelling of workflow components, the basic concept of the ontology was defined as *label type*. Label types can then form hierarchies of type and subtype (similar to class – subclass relationships). Using this model, workflows can then be annotated using *labels*, each of which has a specific label type.

For the implementation of the semantic model the Semantic Web was chosen as basis for reasons of standardisation, tool support, and extensibility.

Ontology creation was made part of the workflow system prototype (Chapter 4 and Chapter 5) to achieve a user experience that is tailor-made for the target application (assisted / automated workflow composition and use). The user interface is described in detail in the next two chapters. Here it should suffice to list the general process of building the ontology.

The most basic concepts – *activity* and *data object* – are predefined. The terms that have been collected in the conceptualisation stage can be added in hierarchical form (taxonomies or paronomies) according to the corresponding grouping. Each term has four fields that should be filled with clear and unambiguous natural language (semi-informal approach):

1. Caption
2. Definition

3. Examples
4. Related terms

The full ontology, with currently over 350 label types, can be found in Appendix A – Label type hierarchies of the developed ontology. More details on user interface, formal language, and storage file format are provided in Chapter 5.

### 3.4.6 Evaluation

METHONTOLOGY suggests to evaluate the created ontology by looking for incompleteness, inconsistencies, and redundancies. As described in subsection 3.2.3, it seems feasible to follow the evaluation criterion based on use cases ([84]) (in line with the PhD methodology in Section 1.4). Chapter 6 lists several examples from the domain of document image analysis, together with an analysis of the application of the developed workflow system (including the presented ontology). During the research, the proposed ontology was used to annotate a large number of software tools (activities with data input and output) and workflows. The ontology was thereby evaluated and modified where necessary. Terms were added, removed, merged, or split to better fit the real world.

## 3.5 Summary

This chapter introduced concepts of representing semantic knowledge. Ontologies are central to this, containing terms and relations of a domain.

Like software projects, ontologies can be engineered using a well-define methodology including a full life cycle. One of the more mature approaches is METHONTOLOGY. It was selected to develop a new ontology for document image analysis. The design steps were described in detail; they included: specification, knowledge acquisition, conceptualisation, integration, implementation, and evaluation.

Two related objectives from Section 1.4 are concluded. Objective 1 (device suitable modelling approach) is satisfied by an ontology-based solution making use of Semantic Web

technology. Objective 2 (semantic model for target domain) is fulfilled by the engineered ontology for document image analysis and recognition.

The next chapter provides details on the design of a prototype workflow system.

## 4 Designing a Semantics-Enriched Workflow System

In Chapter 3, it was decided how to enrich workflow components with semantic data and which ontology representation language to use. Subsequently, a new ontology for document image analysis was engineered. In the framework of the PhD research, the next step is to find ways to use semantic information algorithmically to achieve more automation and to support users in workflow design and management tasks.

This chapter describes the design of a complete workflow system including semantic aspects and algorithms but also general workflow management. The system was then implemented (Chapter 5) and used for experiments and evaluation (Chapters 6 and 7). Only in the context of such a comprehensive system can the usefulness of the semantics-based concepts and solutions be judged satisfactorily.

### 4.1 Workflow Model

The workflow model is the foundation for the whole system. Chapter 2 introduced several existing approaches and modelling paradigms. A decision had to be made to reuse the base model of an established system or create a new one. This also ties in with the implementation phase (Chapter 5). Table 3 provides an overview of advantages and disadvantages of the different options.

The activity-based approach of the ASKALON system [1, 97] was the model of choice due to its extensive documentation, its extensibility, and its foundation in a modelling standard (Activity Diagrams in UML - Unified Modelling Language).

The basic components of workflows (activities, data ports, and data tables) are introduced in the next subsections.

*Table 3 - Advantages and disadvantages of different workflow base models*

<b>Base workflow model</b>	<b>Advantages</b>	<b>Disadvantages</b>
<i>Taverna</i>	Has been used for document image analysis workflows	Several major changes in code basis in relatively short time

<b>Base workflow model</b>	<b>Advantages</b>	<b>Disadvantages</b>
	Source code available	
	Integration with myExperiment	
<b><i>Kepler</i></b>	Mature (based on existing grid technology)	Static execution order
	Source code available	
<b><i>Triana</i></b>	Model includes data types	Less active
	Source code available	Only one data output port per activity
<b><i>Pegasus</i></b>	Only one type of activity	Control flow “hidden” in edges of workflow graph
	Source code available	
<b><i>ASKALON</i></b>	Well-documented	Source code unavailable
	More explicit model (easy to extend)	
<b><i>New model</i></b>	Full control / freedom of design	“Reinventing the wheel”

#### 4.1.1 Concept of Activity

The name “Activity” for processing nodes was chosen by Qin and Fahringer [1] due to correlations to UML Activity Diagrams. Workflows can be expressed and visualised using this standard notation. However, while UML defines *actions* as smallest units, these are not part of the proposed workflow model (where activities are the main components). The proposed concept of an *activity* is consistent with the UML definition: the specification of a parameterised sequence of behaviour. Figure 24 shows the basic notation that will also be used for the remainder of this thesis.

*Atomic activities* are the most basic form of processes. As the name suggests, they cannot be subdivided into smaller units (at least in the context of workflows). If an activity represents a specific existing software tool or method (for instance ABBYY FineReader 11), it is a *concrete* atomic activity and can be executed. If, on the other hand, a generic group of software tools or methods is to be modelled (OCR engines, for example), *abstract* atomic activities are used.

An activity can have input and output parameters which are modelled as *data ports*. These are described in more detail after an introduction to control flow.

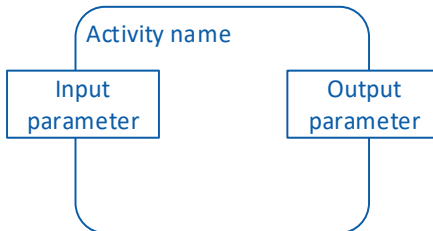


Figure 24 - UML notation of an activity. The activity is represented by a rounded rectangle. Input and output parameters are represented by rectangles on the border of the activity (inputs left or top; outputs right or bottom).

#### 4.1.2 Control Flow

To model complex software processes, more than just atomic activities are required. Qin and Fahringer [1] proposed several types of activities for *control flow*. These are described in the following.

**For Loop Activities** can be used to model iterative processes and are suitable for handling data collections, for instance. Since they only represent control flow, a child activity is required that is itself atomic or contains atomic activities. Four dedicated data ports denote the parameters of a “for loop” as known from programming languages:

- Start position      “for (i=**1**; i≤10; i+=2)”
- End position        “for (i=1; i**≤**10; i+=2)”
- Step width          “for (i=1; i≤10; i+=**2**)”
- Current position    “for (i=**1**; i≤10; i+=2)”

Loop ports are a special case in the workflow model because they are input ports and output ports at the same time. This has two major effects:

- (1) The loop iteration can be controlled from outside the loop.
- (2) Loop parameters can be used as input values for other operations (to access data collections for instance).

If not linked to external data sources, the loop parameters can be specified using the concept of fixed integer values. Normal dataflow is allowed from the loop activity to its child and vice versa (see next subsection for dataflow).

In UML activity diagrams “for loops” can be represented in a generic and a specific form (see Figure 25). Both are so called *expansion regions* or, more generally, *structured nodes*.

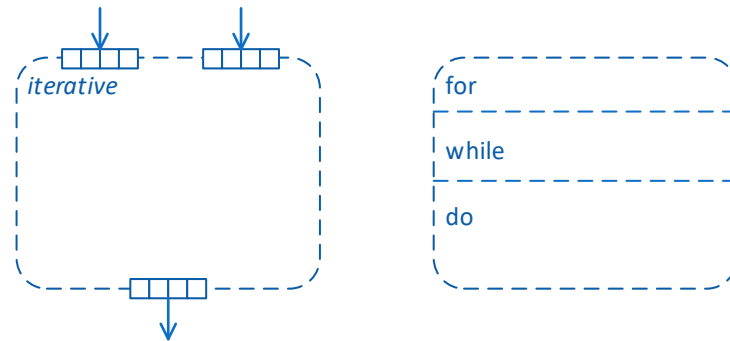


Figure 25 - UML loop notations (left: generic form; right: specific form)

**Direct Acyclic Graph (DAG) Activities** can be used to model more complex control flow. Child activities thereby form a network (a graph) and are executed in a predefined sequence. To avoid endless repetition, cycles are not allowed (*acyclic*). Starting point of execution are child activities that have no predecessor in the graph. The further sequence of processing is defined by the directed edges of the graph.

Parallel execution can be achieved by either having multiple starting nodes or by specifying multiple successors (two or more outgoing edges) at one point in the control flow.

Between sibling activities, dataflow can be modelled by connecting output ports with input ports of the succeeding activity. In addition, the parent activity can pose as data source and data sink.

UML allows nesting of activities. Figure 26 depicts a directed graph activity with three child activities.

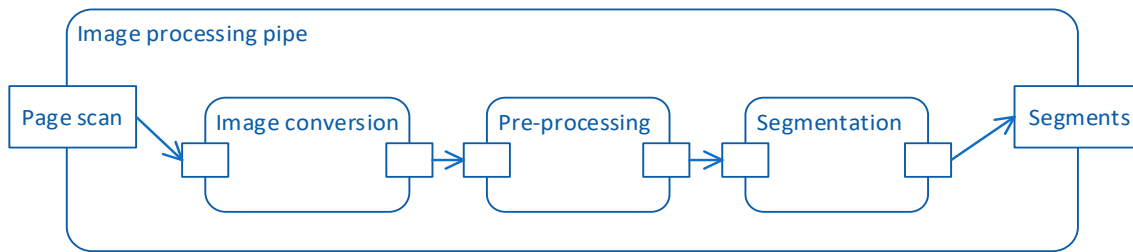


Figure 26 - Nesting of activities in UML (sequences denoted by connected data ports)

**If-Else Activities** represent conditional branches within a workflow. Each branch contains a child activity that is executed if a condition is fulfilled (only the first branch that evaluates to *true*, is executed). A condition can be one or a combination of following condition types:

- Empty condition: Evaluates to *true* (can be used as else-branch).
- Combined condition: Evaluates multiple child conditions, combining them with either AND or OR operation (this is an extension to the if-activity proposed by Qin and Fahringer, allowing for more complex conditions).
- NOT condition: Can have a single child condition, negating its result.
- Input port condition: Evaluates the value of an input port of the if-else activity. Only Integer and Boolean are allowed as data type. Integer values are evaluated to *false* if zero and *true* otherwise.
- Comparison condition: Compares two values (left and right operand) using a selected operator (equals, not equals, less than, less or equal, greater than, greater or equal). The operands can be data objects of input ports or fixed values.

The condition of an if-branch can therefore be represented by a tree of child conditions, allowing for complex constructs.

Figure 27 shows the UML notations of if-else activities (or decision nodes). To achieve a consistent data flow, individual results of the inner activities need to be merged to one single output of the if-else activity (merge node in UML).



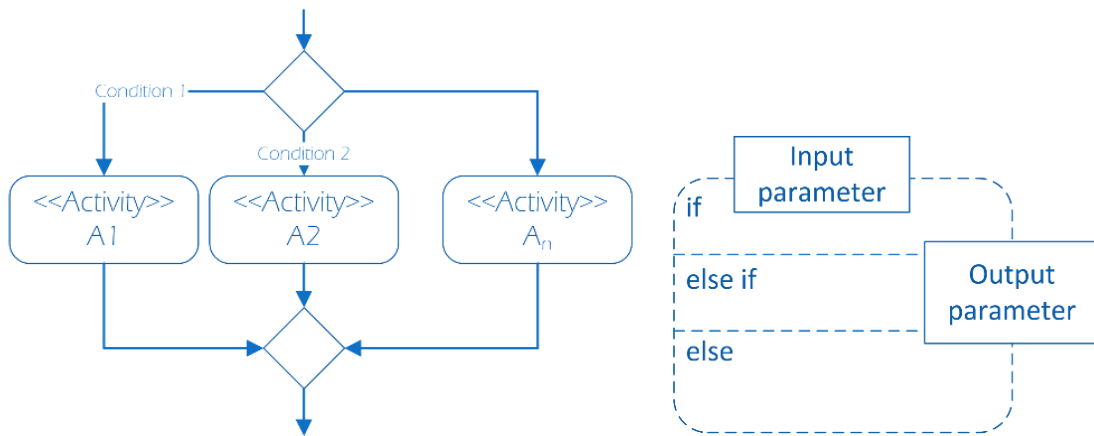


Figure 27 - Decision nodes in UML activity diagrams (left: with fork and merge; right: structured node)

The presented types of activity are sufficient to model complex workflows. Additional types of activities can be added for convenience or clarity. Fahringer and Qin [1] also propose sequence activities (special case of DAG activity) and “while loop” activities.

While control flow determines the general structure of a workflow, dataflow determines how activities are linked to each other. This is described next.

#### 4.1.3 Data and Dataflow

Dataflow is a central aspect of scientific workflows (as mentioned in Chapter 2). The final outcome of an experiment (the result data) is arguably the main interest of researchers. But for reproducibility and integrity, information on data transformations and data provenance in general can be important as well. This subsection explains how the proposed system handles data and dataflow.

##### Forms of data ports

In accordance with UML, all types of activities can have several (or none) *input* and *output* data ports. Most activities have at least one input and one output port (e.g. an OCR activity with image input and text output). Examples of activities without input ports are data creation processes, where the data is generated within the activity itself (for instance a random number

generator). Activities with no output ports are less obvious. Visualisation modules could be categorised as such (having a transient output).

Some types of activities require further data ports. A for loop activity, for example, has four additional *loop ports* (start, end, step, and position).

### **Data types**

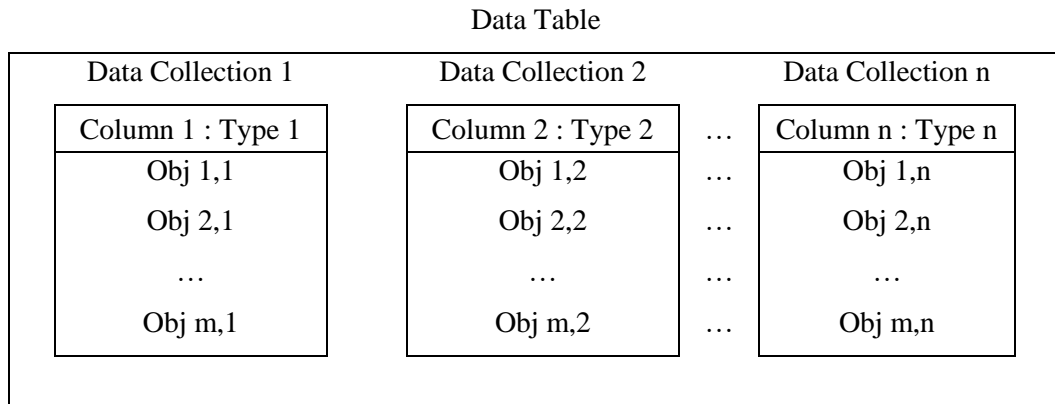
As proposed by Qin and Fahringer [1], ports are assigned either single *data objects* or *data collections* (multiple data objects). A data object has a predefined *data type*. While data types can also represent semantic information, they are lower-level information compared to the semantic layer explored in this PhD research. Data types represent a strict means to establish whether data ports are compatible or not.

Input ports can be assigned multiple types (in contrast to ASKLANON's model where only one type is supported), allowing for a more flexible workflow design (it is common for software tools to support several file formats, for instance). Output ports, however, support only one data type, to enforce deterministic workflows. Otherwise type compatibility could not be checked at design time, only at execution time.

### **Data tables**

Data tables are an extension of the concept of data collections (and are a new concept proposed for this PhD research). Whereas a data collection represents a list of single data objects, a data table represents a two-dimensional matrix of data objects. Each column of a table is represented by a data collection of a specific type. Figure 28 illustrates the structure of a data table.

Data tables can be used as data sources within a workflow or as part of a data repository (see also 4.2). Unlike single data objects or data collections, they are not part of the dataflow of a workflow (i.e. they are not passed around between activities). A workflow can have multiple data tables, all of which can be accessed from each activity of the workflow. The columns of a table resemble output ports of an activity.



*Figure 28 - Model of a data table*

## Dataflow

Dataflow between activities is achieved through *source ports* for input ports and *port forwarding* for output ports. Depending on the type of the activity, a source port can be an input port of the parent activity (for-loop), a sibling activity (graph activity), or any port visible for the parent for instance. Similarly, output ports of a child activity can be forwarded to their parent, for example.

In case of data collections, another form of dataflow is required. By specifying a source port and a port that provides a position within the collection, it is possible to fill the collection with single data objects.

The content of a data object is usually determined when the workflow is executed. However, if primitive types are used (Integer, Decimal, or String), fixed values can be assigned at design time (e.g. “2”, “0.5”, or “blue”).

Figure 29 shows an example dataflow diagram with a for-loop activity and a nested atomic activity. Data tables are global resources within a workflow and can therefore act as input data for any activity. The for-loop has start and end position as inputs and the current position as output. The position is used to retrieve input data items from the table and to add result data items to an output data collection. The atomic activity forwards its output to the parent activity (the loop).

The next subsection introduces abstract workflows (workflow templates).

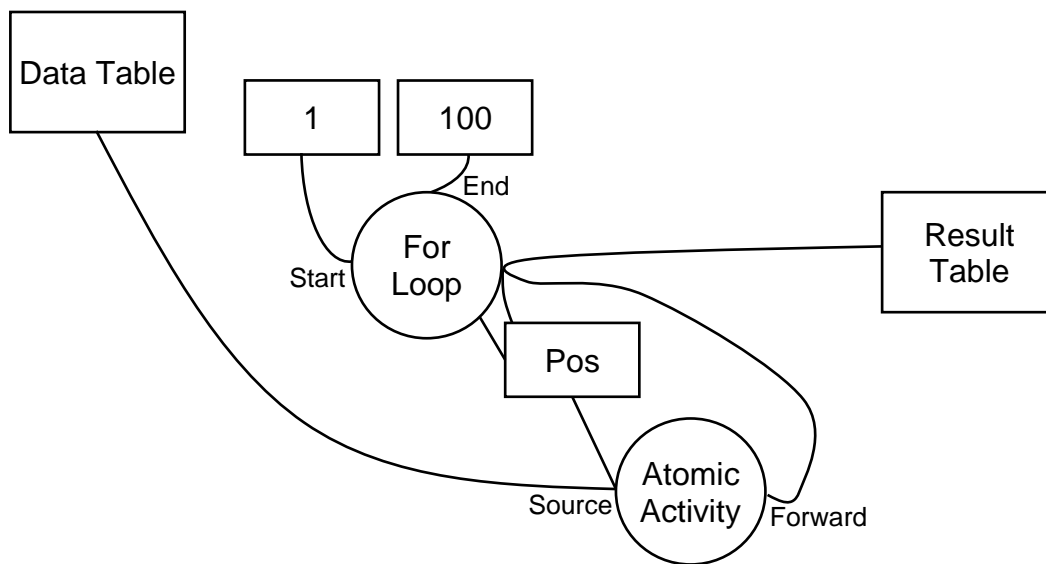


Figure 29 – Example dataflow for two activities

#### 4.1.4 Workflow Templates

*Workflow templates* can be used to model general knowledge about conducting certain types of scientific experiments or constructing production pipelines, without the need to commit to specific software tools or datasets. They are usually created by a domain expert but can then be used by any user for assisted or automated workflow design (see also 2.5.1, 2.5.3, and 2.7).

The concept of templates is realised by means of abstract workflows. Like abstract classes in software design, abstract workflows cannot be instantiated and are therefore not executable. A workflow is abstract if it contains one or more *abstract activities* (Qin and Fahringer use the term *activity function*). Similarly, a control flow activity is abstract if one of its children is abstract. An atomic activity, on the other hand, is explicitly marked as abstract or not abstract (i.e. *concrete*).

An example of a workflow template for performing OCR on a set of scanned document pages is shown in Figure 30. It contains a loop activity with a nested abstract atomic activity. The abstract activity can be replaced by any activity performing OCR (a commercial system and an open source system in this example).

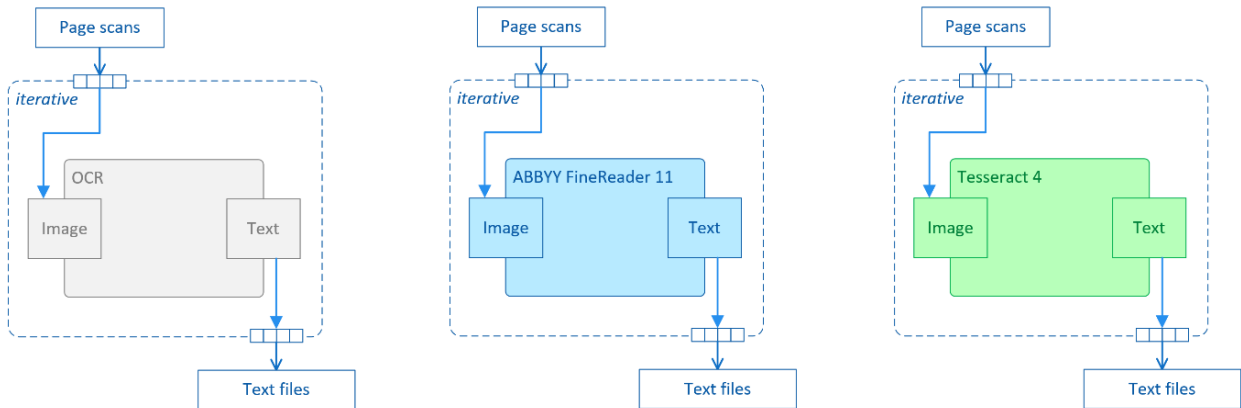


Figure 30 - A workflow template (left) and instantiated (concretised) workflows (middle and right)

#### 4.1.5 Semantic Layer

The concept of semantic labels was introduced in Chapter 3. For the workflow system, this was concretised to three forms: labels, label types, and label groups (this is a completely new concept, not supported in the ASKALON workflow model).

*Label types* can be interpreted as classes. One label type represents a specific feature of an activity or a data object within a certain domain. Types can form hierarchies, which are also called taxonomies or paronomies. A *label group* contains exactly one type hierarchy and defines a cardinality (*label slots*), which can be used to specify how many labels of the group can be assigned to an activity or data object. A *label* can be understood as an instance of a label type (attached to an activity or data object).

It can be summarised that label types form an ontology (the semantic model) and labels are used to attach semantic information to objects of a workflow. Label groups are a supplementary concept with the main intend to improve usability (labels that have been assigned the maximum number of times, as specified by the label slot number, can be hidden from the user).

Within the workflow system, data ports and activities can be annotated with semantic labels from the ontology (see Figure 31). The information can be used for workflow composition and other features.

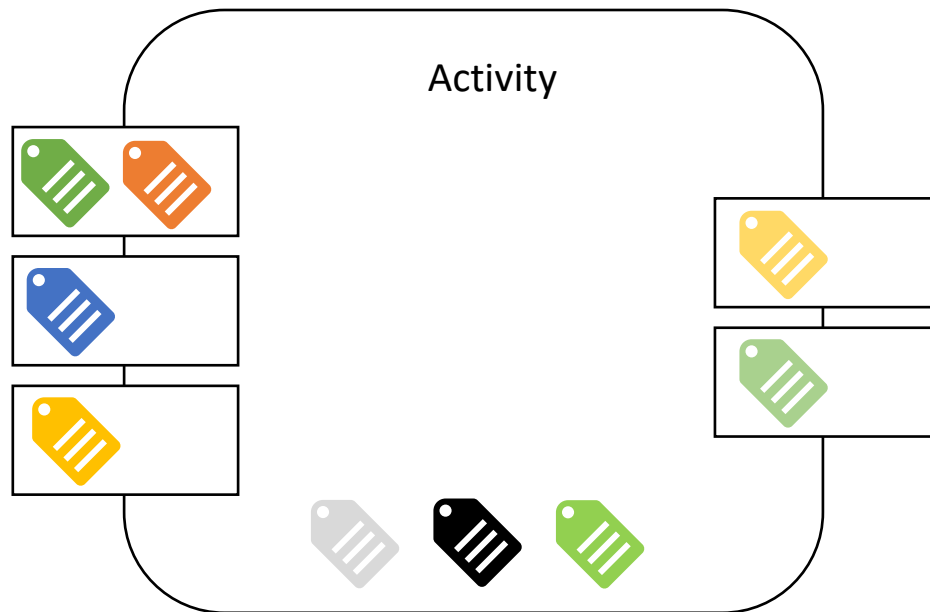


Figure 31 - Illustration of semantic annotation via labels (labels can be attached to data ports and to the activity itself)

#### 4.1.6 User Groups and Perspectives

Different users of the workflow system have varying knowledge of the system and workflow components (see Figure 32 for examples). To improve the usability of the framework in the future, user group and perspective management should be added.

The example ontology was extended by a *User* label slot group, a *User groups* label slot, and a corresponding label taxonomy (see Figure 33). Real-world taxonomies are more complex, but the example is sufficient to describe the respective workflow system features.

Using the ontology extension, each user is then labelled with one or multiple of the user groups (and other label types if defined later). Similarly, certain aspects of the workflow system and workflow components can be labelled, including:

- User interface controls: Specific controls can be hidden from high-level users and only shown to low-level users such as administrators.

- Workflow and activity descriptions: Multiple descriptions can be created, each targeting a specific user group (or groups). To this end, labels can be assigned to the description objects.

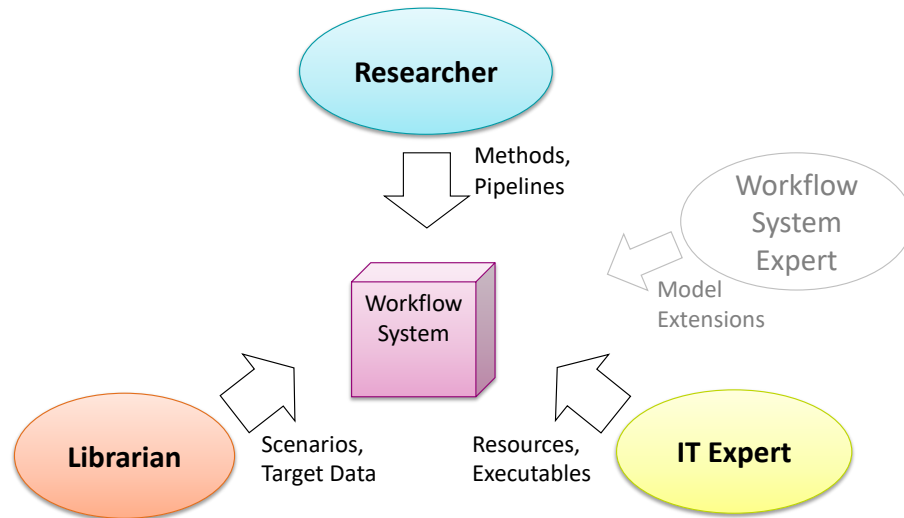


Figure 32 - Example for different views on a workflow system

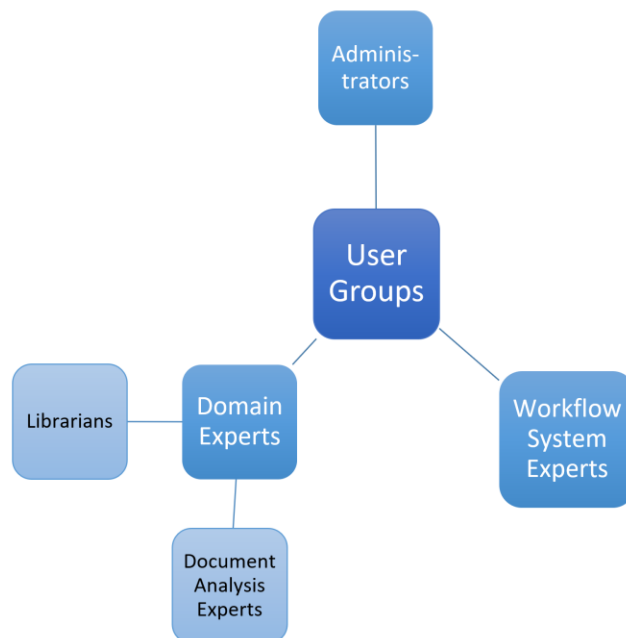


Figure 33 - User groups in the example ontology

## 4.2 Repositories

The concept of workflow repositories was introduced in Section 2.4. For the proposed workflow system, this was extended to three distinct repository types:

- *Workflow repositories*: a collection of workflows and/or templates.
- *Activity repositories*: a collection of concrete atomic activities (representing software tools or services).
- *Data repositories*: a collection of data items that can be used as input for workflows.

In general, the workflow system should provide functionality to create and manage multiple repositories. The following features are considered necessary for the PhD research:

- List and add repositories.
- See content of repository (workflows / activities / data).
- Open item for viewing / editing.
- Add items to repository.
- Remove items from repository.
- Search for items.

## 4.3 Workflow Creation

The proposed system requires a user interface to create and edit workflows. Essential features are:

- Add, edit, remove activities.
- Visualise activities hierarchically (nested activities shown as child activities).
- Add, edit, remove, link data ports.
- Add, edit, remove, link data tables.
- Annotate components semantically (labelling).
- Store and open workflow.



One of the objectives of the PhD research was to discover methods for more automation in the workflow design process. The semantic data, with which activities and data objects can be labelled, can be used to implement automated or assistive features. One such feature is *activity matching*, which is described next.

### 4.3.1 Activity Matching

A major hurdle for workflow designers is finding the right software tool for a certain task. Within workflow design, this manifests in two tasks: (1) adding a new child activity and (2) replacing an existing activity (for instance an abstract atomic activity in a workflow template). Both tasks have in common that an activity must fit into a target environment with certain semantic labels and data types. Within the system, *activity matching* can assist the user in finding fitting activities from a repository.

To be able to identify suitable activity candidates when adding an activity to a workflow or when replacing an existing activity, a matching algorithm based on semantic information can be formulated.

The algorithm can be visualised as a black box, taking activity labels, data port labels, and data types as input and producing a match score as output (see Figure 34). The match score can then be used to sort multiple activities by suitability. An implementation of such an algorithm is described in the next chapter.

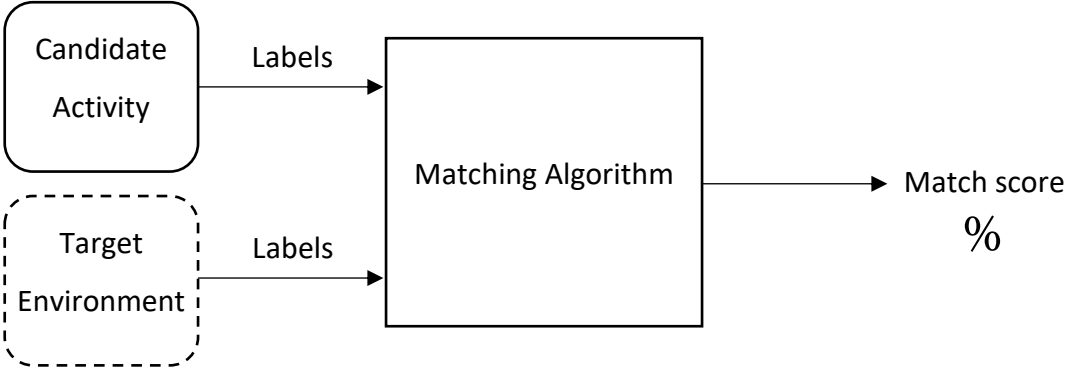


Figure 34 - Concept of activity matching

### 4.3.2 Workflow Validation

The purpose of workflow validation is to improve the quality of workflows by analysing certain aspects and providing the user with feedback. The following aspects should be included in the system:

- Workflow object validation: basic checks if important workflow properties are specified, helping users to find and understand the workflow. In addition, it is checked if the workflow is empty (no root activity) or abstract (not executable).
- Activity validation: Basic checks for activity properties and check if the activity has data ports.
- Missing child activities: checks if there are any activities that should have a child activity but do not have one (e.g. for-loop activity).
- Cycle detection: checks for cycles in acyclic graph activities (cycles are not permitted). A depth-first graph search is performed for each separate set of child activities.
- Unconnected data ports: checks for “loose ends” (input ports that have no source or output ports that are not forwarded anywhere).
- Data type matching: checks that data sources or forwarded data matches the data type of the target port.
- Data cardinality matching: checks that the cardinality of data sources or forwarded data matches the cardinality of the target port (cardinality means single data object versus data collection)
- Missing data types: checks whether all data ports have a data type specified. Ignores abstract activities.
- Label matching: checks that data sources or forwarded data matches the semantic label type of the target port.

The matching and validation represent semi-automated functionality. The next subsection describes the workflow composition features that do not require any user interaction.

### 4.3.3 Automation

One of the objectives of this PhD project was to explore if semantic features can be used to simplify certain tasks within workflow design and use. Complete automation represents the highest achievement in that respect because it would enable non-experts to use the system. While automating every aspect of the workflow design life cycle and execution-related tasks

is beyond the scope of this work, specific sub-tasks can be completed in a fully automated way. These are discussed next.

### **Workflow Concretisation**

As explained in Section 4.1.4, workflow templates are used to model generic tasks or experiments with no commitment to concrete executable activities (software tools / methods). Concretisation is the task of making a workflow template executable by filling in placeholders with non-abstract activities from a provided repository.

Although activity repositories are not defined in the overarching model, they can be easily realised using workflow repositories that only contain ‘dummy’ workflows with exactly one activity (usually an atomic activity).

An algorithm for the workflow concretisation can make use of the activity matching described earlier. Match scores can take into account semantic labels and data types of data input and output ports as well as labels of the activity itself.

### **Data Conversion**

Dataflow is one of the central aspects of scientific workflows. A mismatch of the data formats from one activity to the next will lead to an error at execution time. A workflow system should therefore be able to uncover and, ideally, help to fix the problem at design time.

Data type mismatches can be identified during workflow validation. An automated approach to correct the mismatch could add a conversion activity and re-route the dataflow through the new activity. An algorithm can be developed that replaces a mismatching activity with a directed graph activity containing a data converter and the old activity.

Figure 35 illustrates a conversion for an example, showing a workflow before and after such algorithm was applied.

This concludes the description of the proposed features regarding workflow creation. The next section discusses ontology-related features.

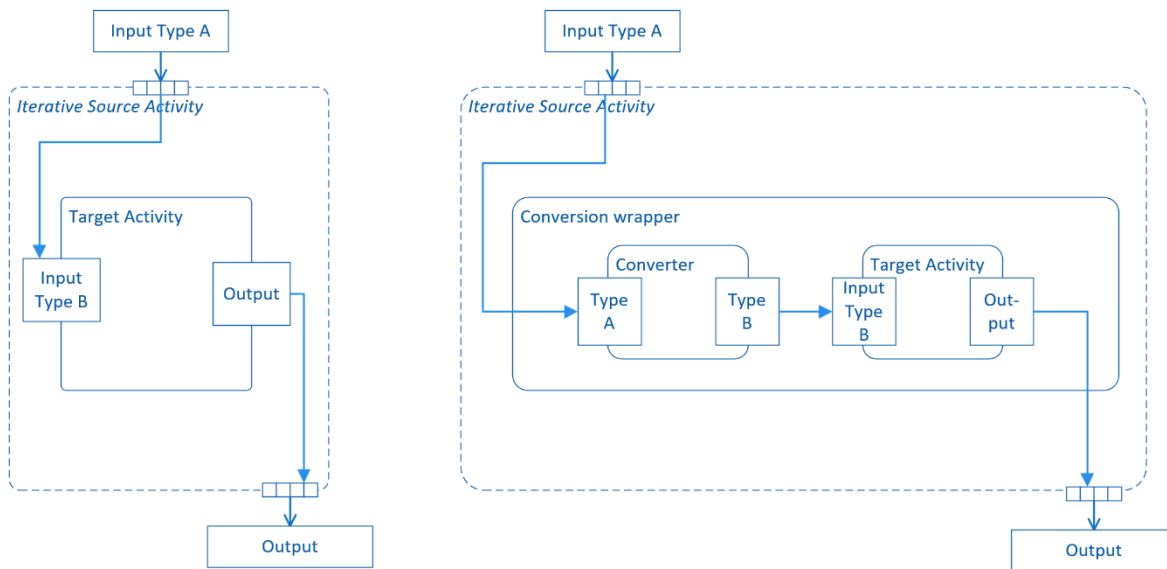


Figure 35 - Data conversion (UML) (Left: before adding conversion step; Right: after adding conversion step)

## 4.4 Ontology Management

Ontologies could be managed entirely by external tools that are based on the Resource Description Framework (RDF) or the Web Ontology Language (OWL) – both are Semantic Web technologies. However, an integrated solution that supports all required functionality in one system is preferable for usability reasons. Specialised tools can hide unnecessary complexities to which users would otherwise be exposed to. The label-based semantic annotation approach only uses a small subset of RDF or OWL. A generic ontology editor would not be limited to that subset.

An ontology editor should provide the following functionality:

- Browsing (for use during workflow composition).
- Editing (for creating and maintaining ontologies).
- Storing (for local use and sharing).
- Migration and versioning (for ontology life cycle management).

## 4.5 Summary

A workflow model based on activities and data connections was presented (following the UML Activity Diagram concepts). The design is inspired by ASKALON's workflow paradigm but was heavily adapted to satisfy the needs of this PhD research. The extensions include data tables, semantic annotation using labels, a matching algorithm, and user-related features.

Objective 3 (semantics-based algorithms for automation and supporting users) from Section 1.4 can be seen as fulfilled by the presented algorithms for activity matching, workflow concretisation, and assisted data conversion. Objective 4 (a framework for workflow creation/management using the ontology and algorithms) is partially fulfilled (design stage complete, implementation stage described next).

The proposed design should enable assistive features to help users to create workflows. In order to test and evaluate this, a working system is required. The next chapter discusses the implementation of such a system.

## 5 System Implementation

This chapter provides a detailed description of how the workflow system proposed in Chapter 4 was implemented. The physical realisation of a comprehensive workflow framework (with semantics-enabled functionality) is key to test the hypothesis that the incorporation and algorithmic use of semantic metadata can assist users in workflow design and related tasks.

To facilitate future reuse and integration, Java was selected as programming language. All systems described in Chapter 2 are Java-based and could make use of the semantic features and algorithms.

Being developed as a proof of concept, the new system requires the following core components:

- Ontology editor (to create and maintain an ontology).
- Workflow composition tool (to create example workflows and test semantics-based features).
- Workflow repository (to test semantic search and provide input for workflow composition).

The prototype was developed to a state where it is fully functional regarding the core components. A complete workflow system typically has additional features that were omitted for the prototype (because they are irrelevant for the PhD research). These include:

- Workflow scheduling and execution.
- External tool integration (web services or local processes).
- Data provenance functionality.
- Failure handling.
- Distributed design (remote resources).

The software development was conducted using the Rapid Application Development (RAD) method [98] because it suits the evolving nature of the PhD project. The approach contains the following main principles (inapplicable sections left out):

1. “Key objective is for fast development and delivery of a high-quality system at a relatively low investment cost.”
2. “Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more easy-of-change during the development process.”
3. “Aims to produce high quality systems quickly, primarily through the use of iterative prototyping (at any stage of the development) [...] and computerized development tools. These tools may include Graphical User Interface (GUI) builders [...] and object-oriented techniques.”
4. “Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.”
5. “Project control involves prioritizing development and defining delivery deadlines or ‘timeboxes’. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not increasing the deadline.”
6. “Generally includes Joint Application Development [...]”
7. “Iteratively produces production software, as opposed to a throwaway prototype.”
8. “Produces documentation necessary to facilitate future development and maintenance.”
9. “Standard systems analysis and design techniques can be fitted into this framework.”

With respect to point 3, the default designer tool of the Eclipse IDE (integrated development platform) was used, which produces Java Swing user interface components. In addition, the software design included standard object-oriented concepts such as inheritance, templates, and interfaces as well as design patterns (e.g. Observer pattern, Factory pattern and Singleton).

In the following, system components will be illustrated using simplified UML diagrams of selected interfaces and classes. Extended diagrams and a full implementation list can be found in Appendix B – List of All Implemented Interfaces and Classes.

## 5.1 System Overview

The prototype system is comprised of three main software components, each with graphical user interface (see Figure 36):

- (1) *Ontology editor*: used to create and edit ontologies by defining label type hierarchies.
- (2) *Workflow repository hub*: offers functionality to create and manage collections of workflows (repositories) as well as list and search for workflows. The repository hub is linked to a specific ontology (here the ontology for document image analysis).
- (3) *Workflow editor*: is used to create or edit workflows. The editor is a standalone tool but can also be instantiated by the repository hub. Like the hub, the workflow editor (and the current workflow) is linked to the ontology of choice to provide the user with available label types (when adding semantic information to workflow components).



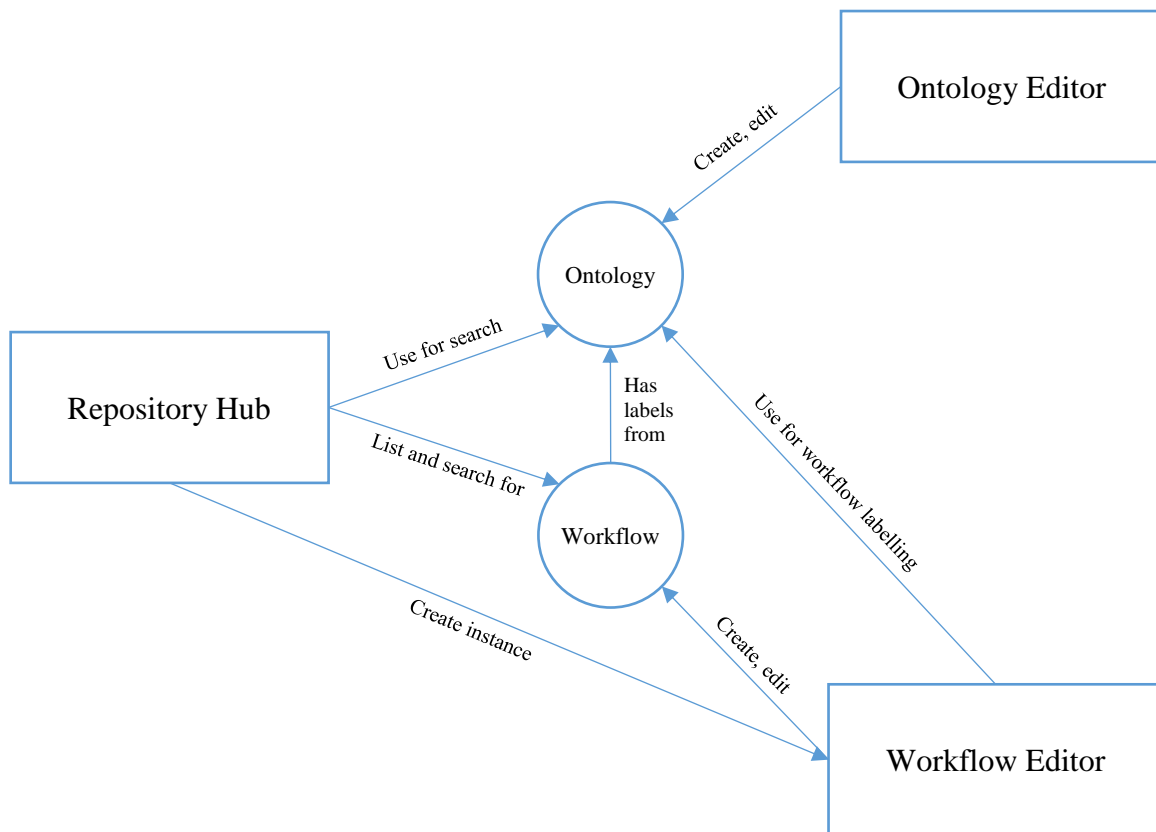


Figure 36 - Architecture of prototype (rectangles: standalone software tools; circles: central objects)

The system design from Chapter 4 was translated to Java classes and interfaces in order to facilitate the desired functionality. An overview of the key elements is provided next.

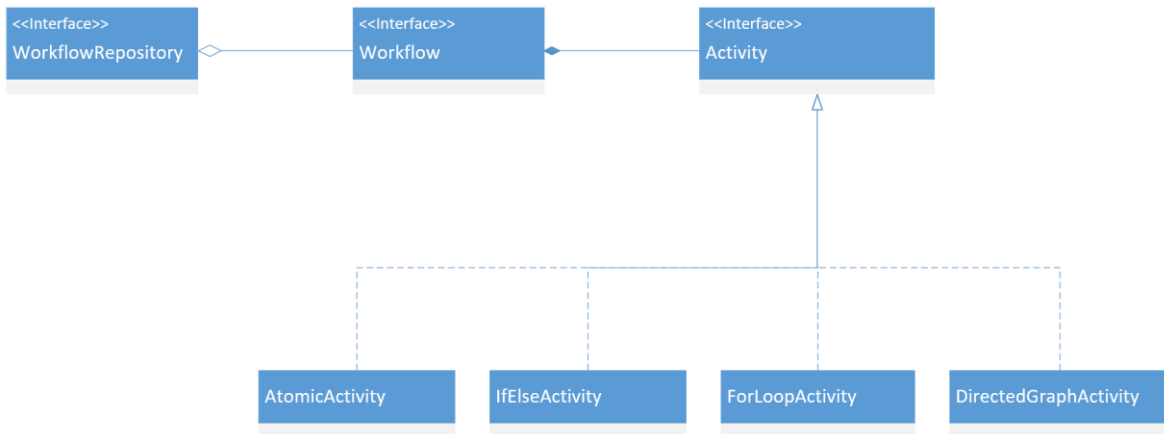


Figure 37 - UML Diagram for repository, workflow, and activity

The main classes and relationships of the proposed workflow model can be seen in Figure 37. A *WorkflowRepository* is therein composed of *Workflow* objects, which is in turn composed of *Activity* objects. The four essential types of activities (Section 4.1) are modelled as child classes of *Activity*.

Semantic metadata is integrated via *Label* objects. An ontology thereby defines groups of label types. Activities and data objects can then be tagged with specific labels from within these taxonomies (Figure 38).

This concludes the system overview. The next sections provide details on user interface, class structure, and data formats used for the prototype.

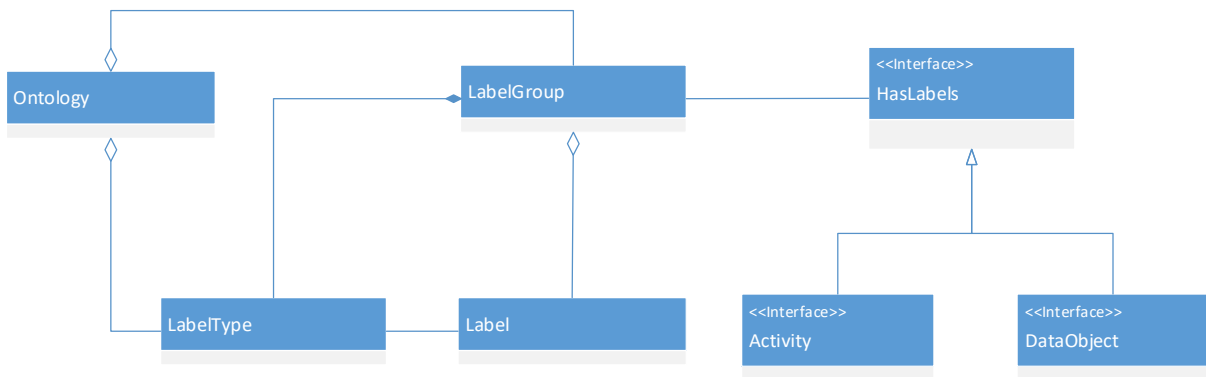


Figure 38 - UML diagram for ontology and labels

## 5.2 Ontology Editor

The concept of semantic labels was introduced in Chapter 3 and refined in Chapter 4. For the prototype implementation this led to three elements: label types, label groups, and label instances (see again Figure 38). Implementing a dedicated ontology editor has the advantage of being able to confront the user only with relevant features and hide other complexities of the Semantic Web (i.e. concepts that are not required for the chosen modelling approach).

The following subsections introduce ontology-related features of the workflow system.

### 5.2.1 User Interface

The ontology editor (Figure 39) provides a graphical user interface for building and maintaining ontologies. It includes the management of taxonomies (label groups) and their assignment to labellable objects using a cardinality (label slots).

Label types are visualised in a tree view, organised by taxonomy (root label type; see Figure 39 top left). Apart from a necessary ID, label types can also contain informational metadata (see Figure 39 top right).

A taxonomy can be assigned to one of the predefined labellable objects (here activity or data object; see Figure 39 bottom). Label slots define the maximum number of labels (per taxonomy) that can be assigned to an object (i.e. the cardinality).

As mentioned in previous chapters, an ontology is an ever-evolving construct. Similar to software systems, a versioning approach is crucial. The editor allows the specification of an

ontology version number as well as the definition of rules for migrating from an older ontology to the current one (see subsection 5.2.3).

As supplementary feature, the export of a label type hierarchy as a table in comma-separated values (CSV) format was added. This was used to create the table in Appendix A – Label type hierarchies of the developed ontology.

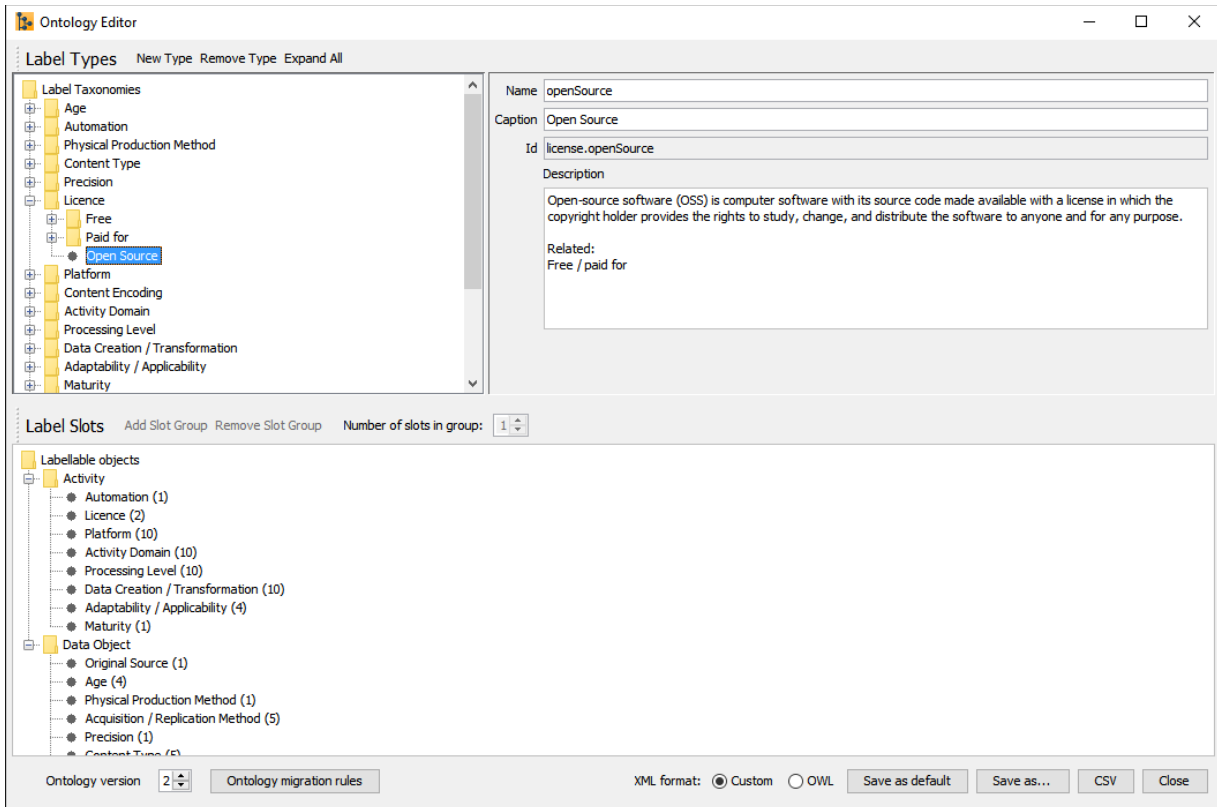


Figure 39 - Ontology editor

## 5.2.2 Ontology Data Format

Due to the limited set of basic concepts of the ontology (label types / groups for activities and data objects) and because it is straightforward to encode the hierarchical structure in XML, a dedicated XML-based data format was specified (as opposed to adopting an existing data format). Reader and writer classes were added to the prototype system. Listing 1 shows a shortened file content of an example ontology.

Using a dedicated format improves the readability of files and the robustness of the system. A disadvantage is that only specialised software tools can work with data stored therein. An alternative would be of interest for future versions of the system. The concept of labels, label types, and label groups can also be expressed using the Web Ontology Language (OWL) standard [4]. Established editors (such as Protégé [60]) could be then used to create ontologies. Section 5.5 provides details of the translation approach for ontologies and semantic data of workflow components.

*Listing 1 - Ontology XML format (example)*

```
<Ontology version="2">
  <LabelTypeHierarchies>
    <LabelType caption="Domain" name="domain">
      <Description>...</Description>
    <LabelType caption="Document Image Analysis" name="dia">
      ...
    <ActivityLabelSlots>
      <LabelSlotGroup name="processing-type" slots="3"/>
      ...
    <MigrationRules version="1">
      <SourceType id="processing-type.acquisition">
        <TargetType id="dataTransformation.acquisition"/>
      ...
    </MigrationRules>
  </LabelTypeHierarchies>
</Ontology>
```

### 5.2.3 Ontology Migration Rules

Small changes of an ontology, such as extending a label type taxonomy, do not require any migration solution because no inconsistencies within existing resources can occur. As soon as the hierarchy structure is changed however, existing data may become invalid. There are two options to deal with this problem: disallow the handling of data that has been created using a previous ontology or enable an implicit migration to the latest version.

The second solution can save time, if a considerable amount of workflow data has been semantically annotated already. But even using an automated approach, a manual inspection of the migration result is still recommended.

A rule-based method was implemented which uses explicit transformation rules that represent a one-to-many relation. A (source) label type of a previous ontology can be associated with targets from the current ontology, adhering to one of the following options:

- No target label type: the label will be deleted.
- One or multiple target label types: the label will be replaced.

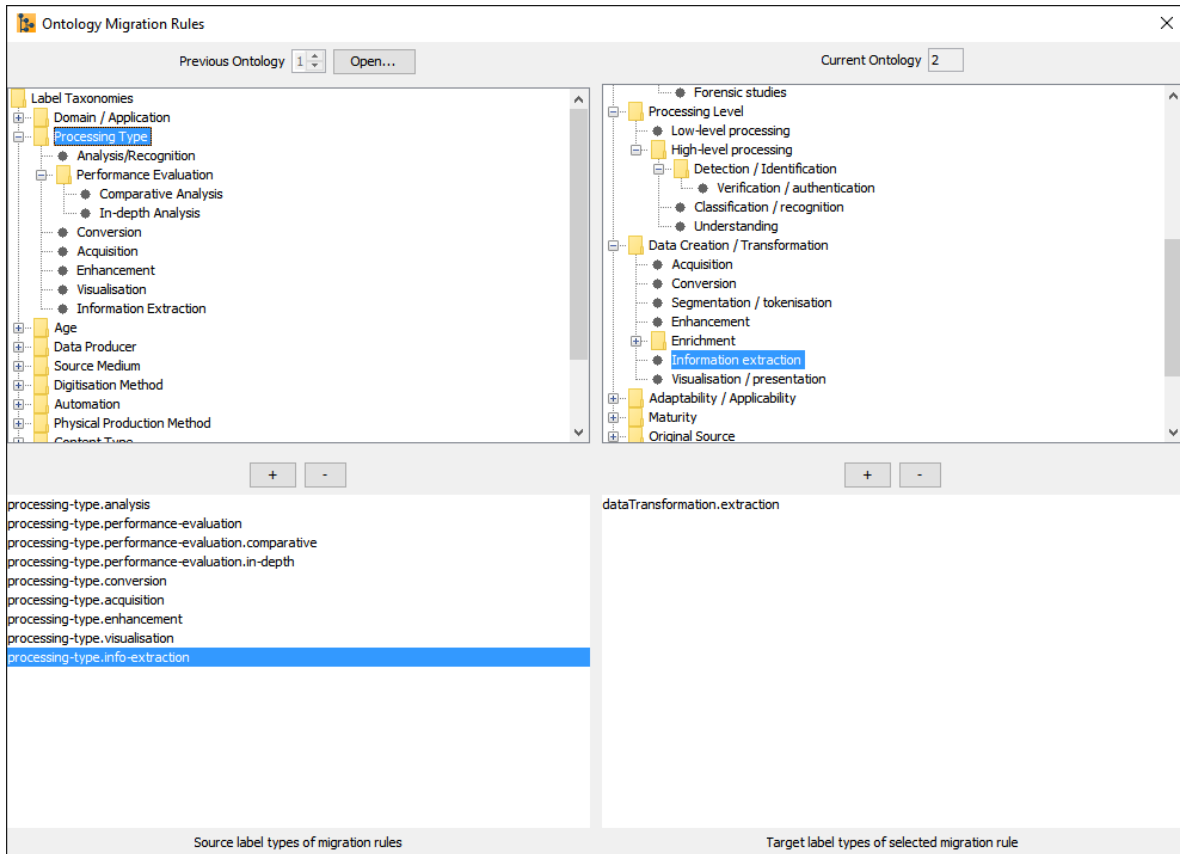


Figure 40 - Ontology migration rules dialogue (top left: version number and label type tree of an older version of the ontology; top right: version number and label type tree of the latest version of the ontology; bottom left: list of existing migration rules for old to new version (and controls to create or delete rules); bottom right: target label type(s) of selected migration rule (and controls to add or remove label types)

Figure 40 shows the dialogue that is used to define the migration rules (accessible from the ontology editor). The rules are saved as part of the current ontology. The user interface provides two label type trees for a previous ontology (selected by the user) and one for the current ontology respectively. Migration rules are listed by their source label types (bottom

left). The associated target label types for the currently selected source type are shown on the bottom right.

This concludes the description of the ontology editor. The next section provides details on features related to repositories.

## 5.3 Repository Hub

In this section, workflow repositories (introduced in Section 2.4) and data repositories are discussed in more detail. Although they represent complementary functionality (the workflow system can be operated without), repositories increase the usability by making existing resources easily accessible. The resources include: full workflows, partial workflows / activities, and data objects (single data items, collections, or data tables).

### 5.3.1 Workflow Repositories

Figure 41 shows the hierarchy of the most relevant classes from the prototype application (see Appendix B – List of All Implemented Interfaces and Classes for an extended diagram). A repository therein represents a simple collection of workflows. Additional functionality, such as search, is realised through associated classes (e.g. filters).

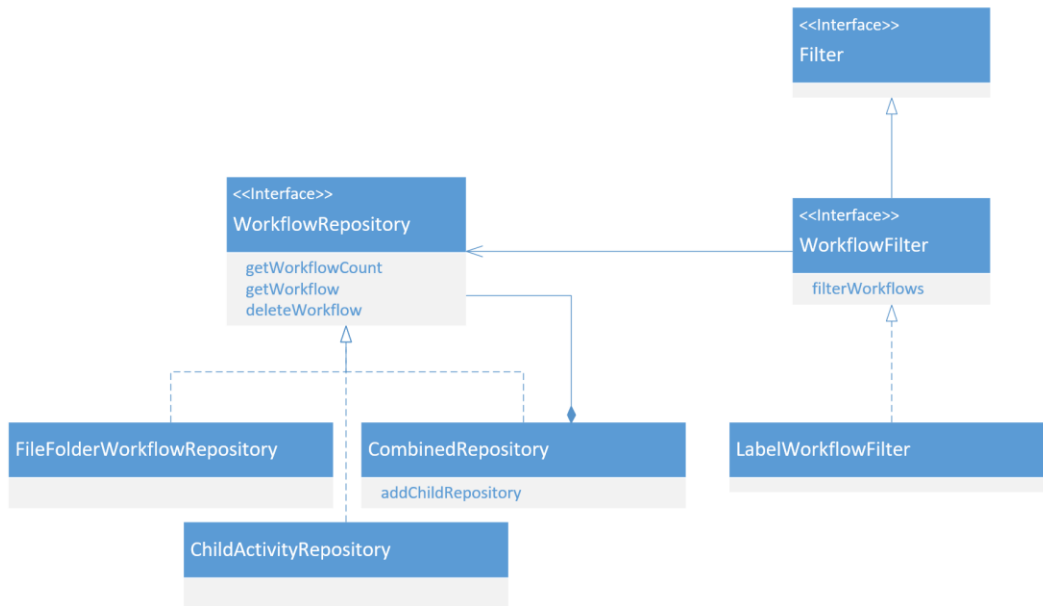


Figure 41 - UML diagram for workflow repository and filter

An instance of the class *FileFolderWorkflowRepository* is linked to a folder of the local file system and aggregates all workflows that are stored within (as XML files). A *CombinedRepository* can contain several child repositories and can be seen as a meta repository. A *ChildActivityRepository* lists all child activities of a given workflow as individual workflows. Future systems could encompass more complex repositories using databases or web services.



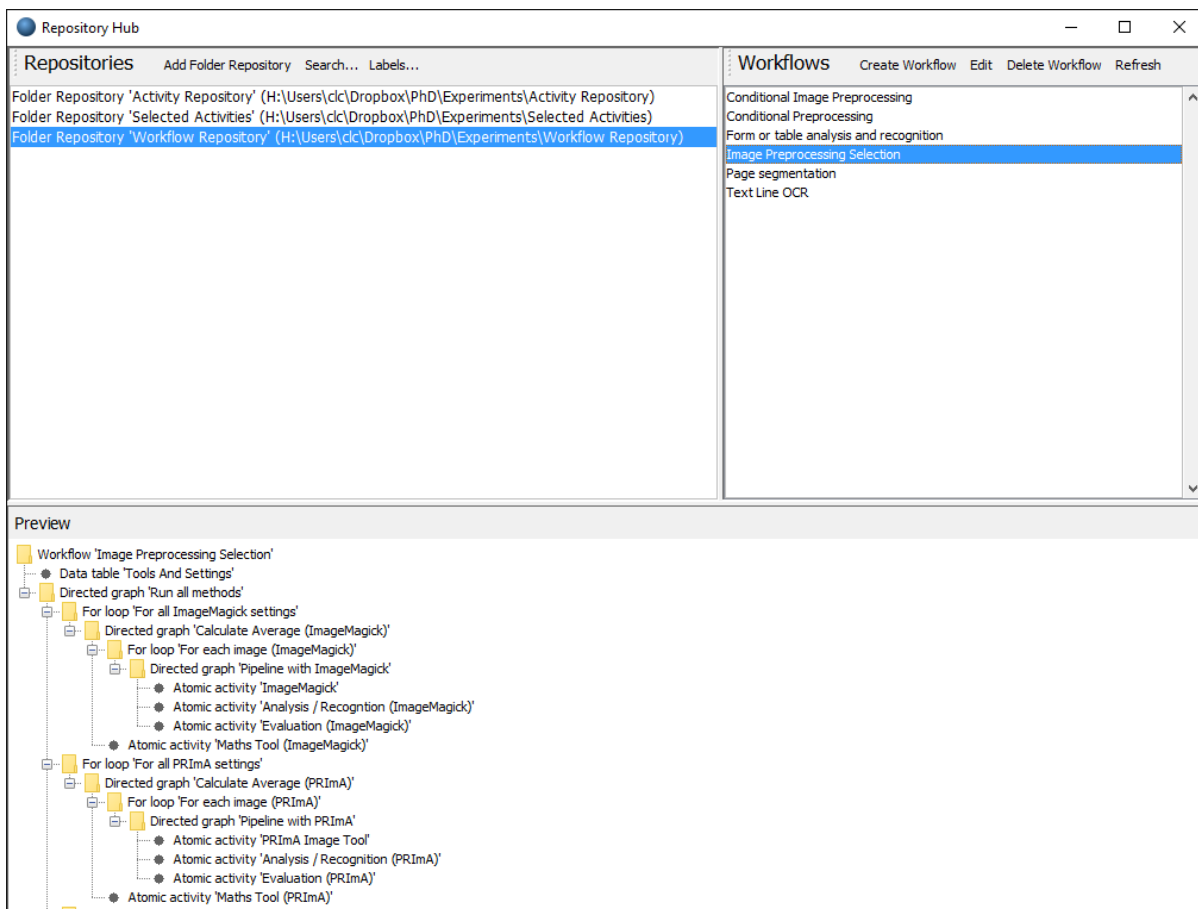


Figure 42 - Repository Hub

A software tool called “Repository Hub” (see Figure 42) provides an entry point into the workflow management of the prototype system. The user interface implements a master-details view design, with a list of workflow repositories on the left (master) and the workflows of the selected repository on the right (details). A preview of the currently selected workflow loop is shown at the bottom. The following features are provided:

- List and add workflow repositories (Figure 42 top left).
- See content of repository – workflows (Figure 42 top right).
- Preview workflow (Figure 42 bottom); open workflow for viewing / editing.
- Add / delete workflows (Figure 42 top right toolbar).
- Workflow search (Figure 42 top left toolbar).

Figure 43 shows the workflow search dialogue. The main search functionality is realised using a cascade of filters, each calculating a subset of workflows based on their current setting. In addition, the search results can be refined using text search (in workflow and activity metadata) and an option to only return concrete (i.e. non-abstract) workflows. Adding more filters in the future is straightforward (a data type filter for workflow inputs and outputs, for instance).

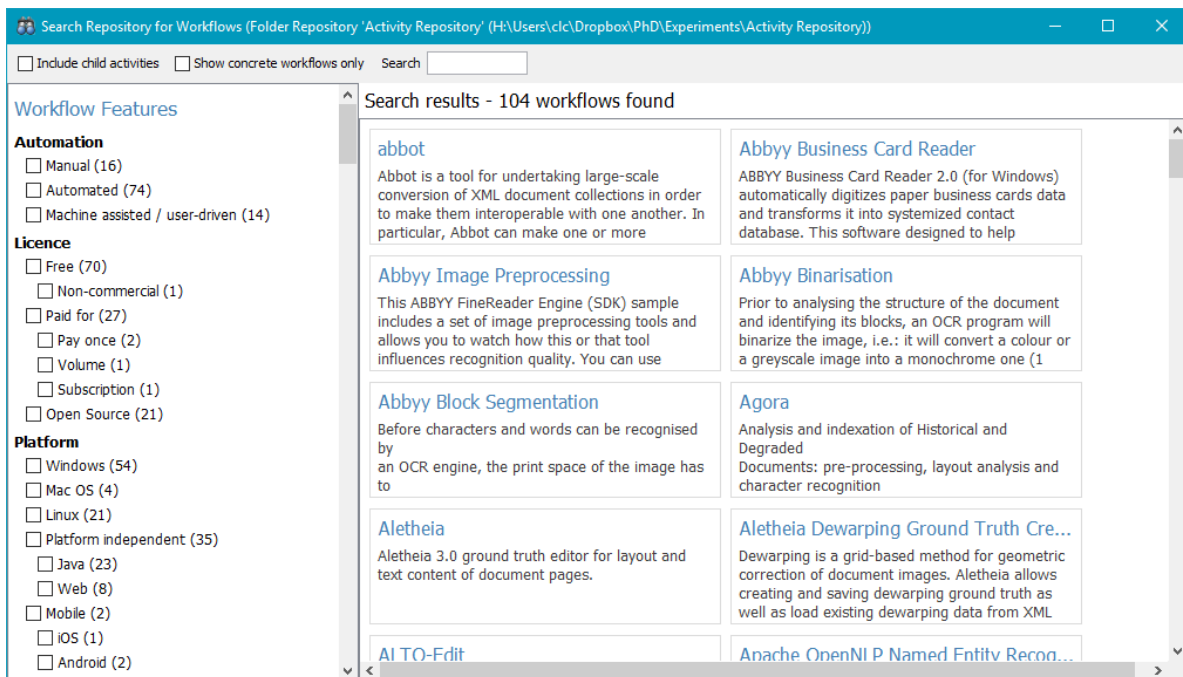


Figure 43 - Workflow search interface

The label-based search is controlled via checkboxes – one for each label type. The domain-specific ontology (see Section 3.4) is thereby used to obtain a list of all possible labels. Unused label types (types not used in any workflow), however, are not shown in the user interface. Numbers next to the checkboxes indicate how many workflows are labelled by the corresponding type. Once the user ticks a checkbox, only workflows with the specific label are shown in the search result. The numbers for the boxes are updated accordingly.

Since a workflow object itself does not have semantic properties in the proposed model, the root activity of the workflow and its input and output data ports are used instead. In future implementations this can be extended to all activities within a workflow (optionally).

The search result shows title and description of the found workflows. Further details on input and output ports are available via tooltips. A mouse click on a result item opens the respective workflow in the Workflow Editor (see next section).

### 5.3.2 Data Repositories

Like workflows and software tools (i.e. atomic activities), datasets represent an important resource for research and development. For this reason, data repositories were added to the design (Section 4.2).

As mentioned in Section 4.1.3, workflows can contain data tables. A data repository is defined as the sum of all data tables of a workflow repository. This design reduces the development effort because several core and user interface components can be used for both workflow and data repositories. The workflows are not required to contain activities and can therefore be pure data sources.

Search functionality for data tables is integrated as part of the workflow search dialogue described in the previous subsection. Checkboxes for labels of table columns can be found at the at the bottom of the filter panel (see Figure 44).

Data tables can be enriched with semantic labels. They can therefore be used for the automation of workflow composition by providing relevant input data (example sets, tool settings etc.). The prototype system only supports data tables with static content, but future extensions can include dynamic implementations that link to databases or other sources.

More information on data tables can be found in the next section, which provides details on workflows and related components.

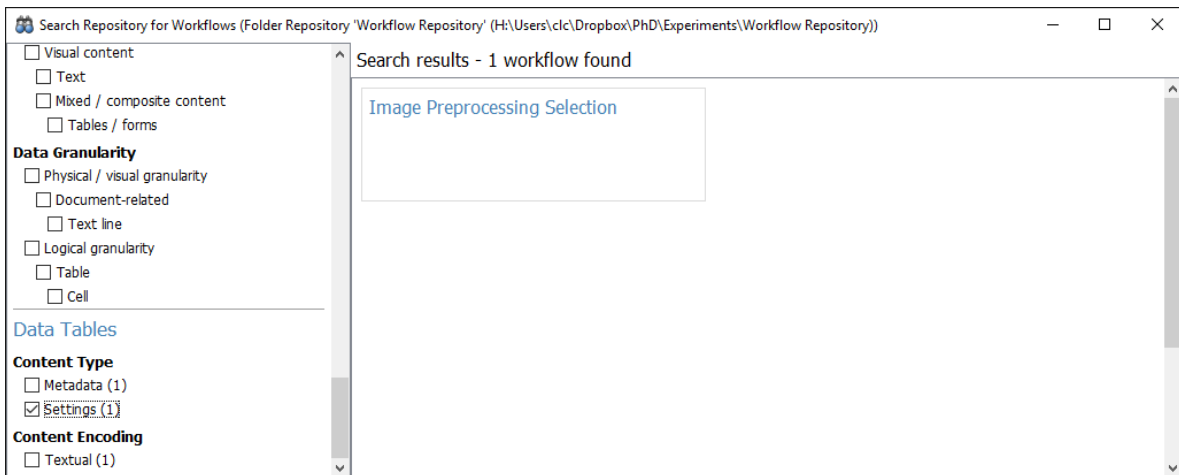


Figure 44 - Searching for data tables within a repository

## 5.4 Workflow Editor

A workflow is composed of activities, of which one is called *root activity*, representing the entry point of the workflow. Activities can be classed into atomic activities and control flow activities that can have child activities (see Section 4.1). The set of all activities can be represented by a tree structure, with atomic activities as leaf nodes.

The Workflow Editor (Figure 45) was implemented to either work stand-alone or in conjunction with the Repository Hub. The main window shows the activity tree on the left side and details of the currently selected item on the right side. It should be noted that the root of the tree within the editor represents the workflow object itself and not the root activity, which is represented by the (only) child item of the workflow node.

The workflow object entails basic metadata such as name, version, author, and description. Activity data is split into general data (ID, caption, description, semantic labels, data ports, etc.) and specific data that is different for each of the activity classes (if-else, for loop, graph, atomic).

The following subsections provide details on the different aspects of workflows as implemented within the prototype system.

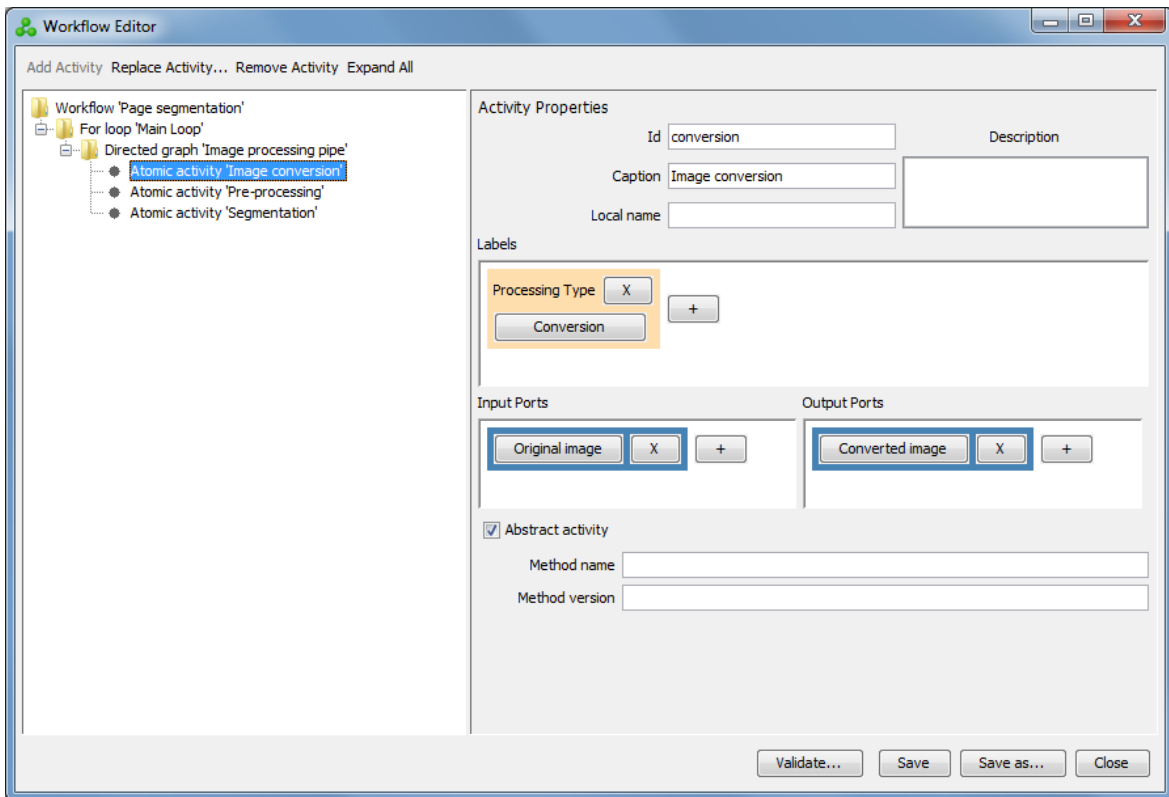


Figure 45 - Workflow Editor (left: tree view of activities; right: details of selected activity)

### 5.4.1 Data Ports

Dataflow and data objects were discussed in Section 4.1.3. This subsection explains how data-related concepts were implemented.

Dataflow is modelled through *DataPort* and *DataObject* (see Figure 46). Each activity can have several input and output ports. Data objects can take the form of *SingleDataObject* (for instance an Integer value or an image file) or a *DataCollection* (i.e. a list of data objects).

Figure 47 shows the data port dialogue of the workflow editor. Data ports can and should be enriched with a caption, a description, semantic labels, and data types (see below). An ID is designated automatically.

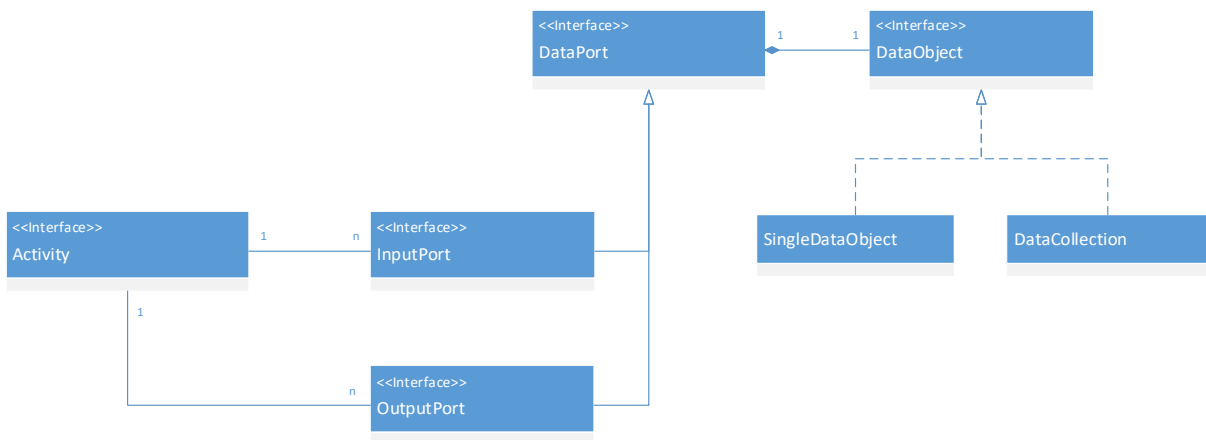


Figure 46 - UML class diagram for data ports and data objects

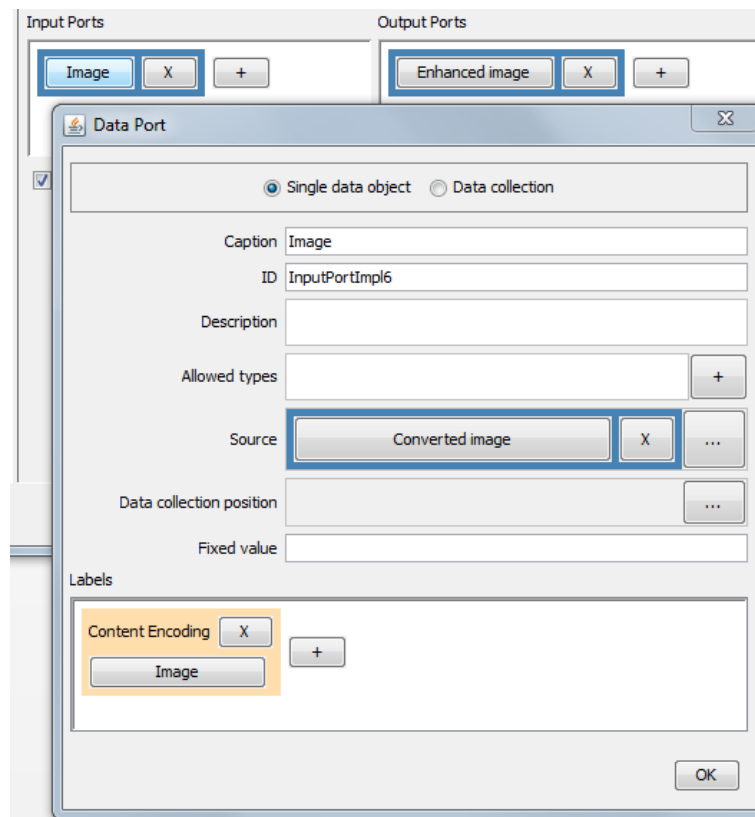


Figure 47 - Data port dialogue

## Data types

The prototype implementation uses a basic data type system based on a pre-defined type hierarchy (stored in a specific XML-based format).

Input ports can be assigned multiple types. Output ports support only one data type. Figure 48 shows the data type selection dialogue. The usage of a predefined type is thereby not enforced; generic types can be chosen as well. Type equality is determined by the IDs of the specified data types.

If primitive types are used (Integer, decimal, or string), data objects can be assigned fixed values at design time.

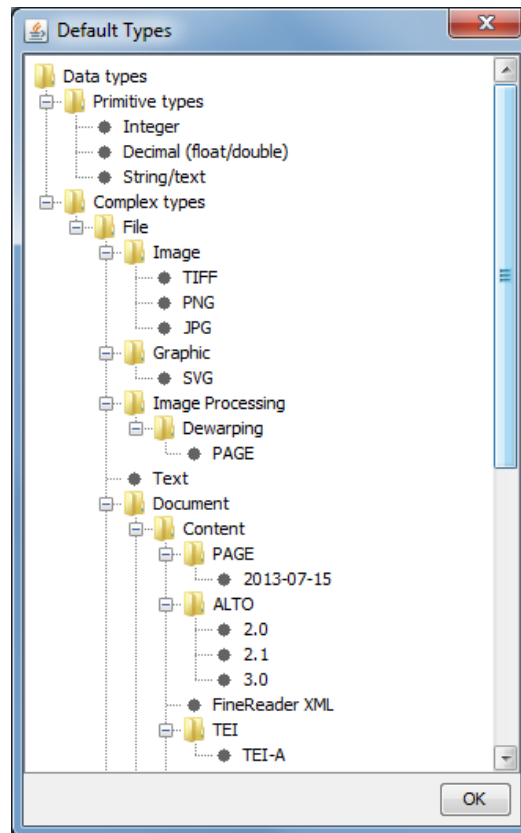


Figure 48 - Data type selection

## Data connections

Dataflow between activities is achieved through *source ports* for input ports and *port forwarding* for output ports. The workflow editor determines automatically what ports are available as source or for forwarding and only presents these to the user (see Figure 49 for an example).

In case of data collections, another form of dataflow is required. By specifying a source port and a port that provides a position within the collection, it is possible to fill the collection with single data objects. The type of the position provider port must be “Integer”.

The next subsection provides implementation details on workflow activities.

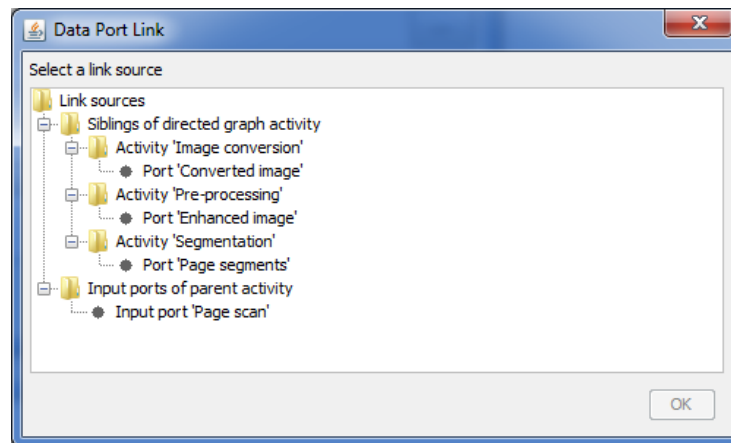


Figure 49 - Selection of input port source

### 5.4.2 Activities

This subsection provides implementation details of the different activity types introduced in Section 4.1.1 and Section 4.1.2.

*Atomic Activities* represent a software tool or service. They can also be used as a placeholder within workflow templates. The workflow editor denotes this explicitly through a tick box (“abstract”) within the workflow details panel. Figure 50 shows a concrete activity in the workflow editor.



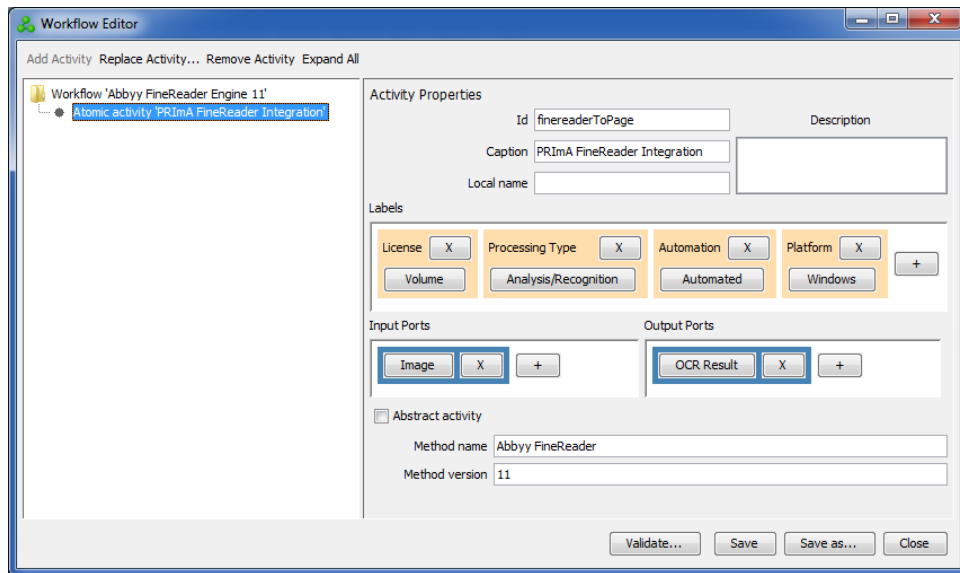


Figure 50 - Details of a concrete atomic activity (method name and version are provided and “Abstract activity” is not checked)

For Loop Activities model iterative processes. Each loop activity needs to have exactly one child activity (of any type). Figure 51 shows the workflow editor view of a loop activity. Four *loop ports* are used to specify start position, end position, step with, and current position. Loop ports are input ports and output ports at the same time (see Figure 52).

If not linked to external data sources, the loop parameters can be specified using fixed integer values, as described in the previous subsection. Normal dataflow is allowed from the loop activity to its child (via source port definitions) and vice versa (via output port forwarding).

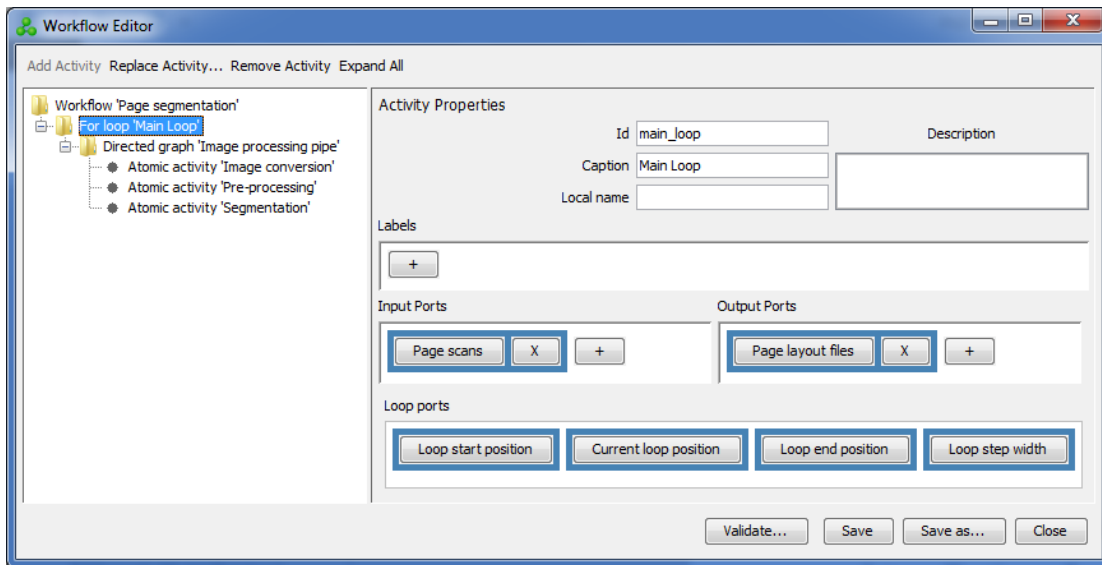


Figure 51 - Details of a “for loop” activity (special loop ports at the bottom right)

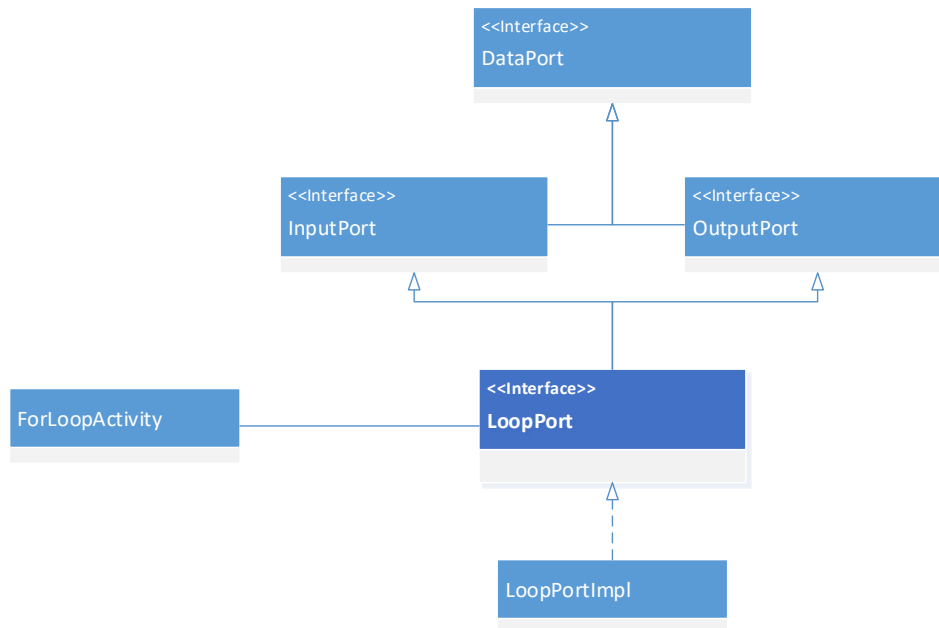


Figure 52 – UML class diagram containing loop port

*Direct Acyclic Graph (DAG) Activities* can be used to model sequences and parallel processing. Figure 53 shows an example of a simple non-parallel pipeline of three activities. The nodes of the graph (representing the child activities) can be freely arranged by the user. The layout is saved in the workflow XML data file.

Parallel execution is achieved by using multiple start nodes or by specifying multiple successors at one point in the control flow. A special concept thereby are optional execution branches. These are reserved for (abstract) workflow templates and represent processing paths that need not be used in a final (concrete) workflow. Furthermore, a minimum number of successors can be defined for an activity, to indicate how many of the optional paths must be realised (for the workflow to be valid). One example is a workflow that compares the performance of two or more methods. Other use cases for this feature are provided in the next chapter.

Dataflow is created by connecting output ports with input ports between activities.

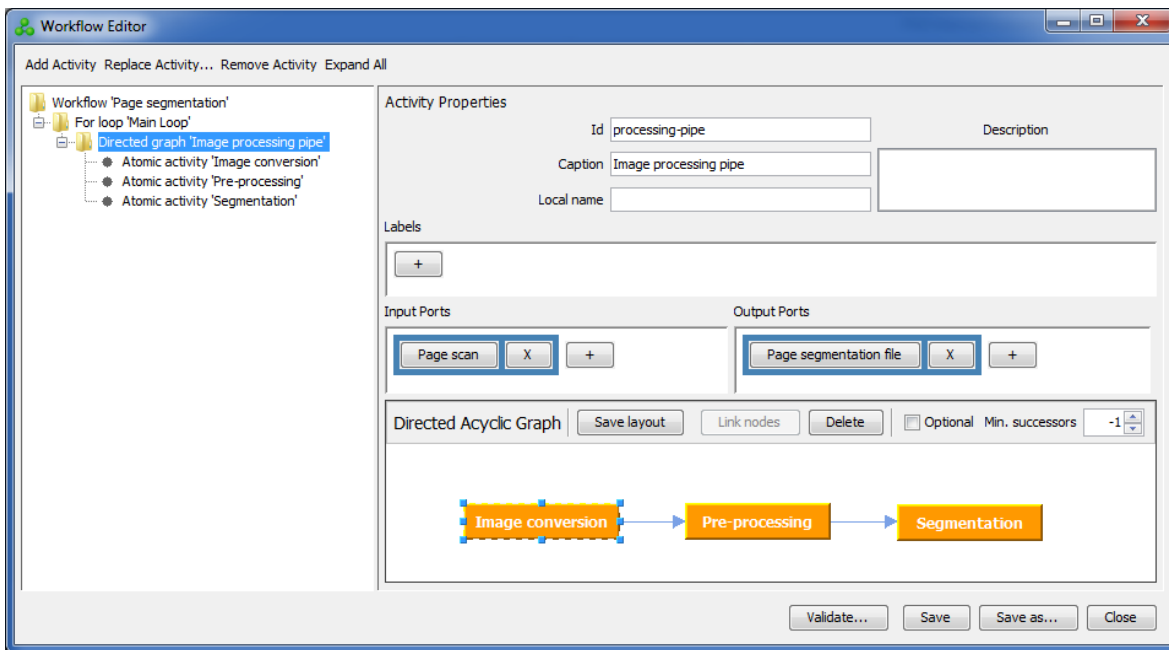


Figure 53 - Details of Directed Acyclic Graph activity in Workflow Editor

*If-Else Activities* represent conditional branches. Figure 54 shows an example in the Workflow Editor. The specialised panel on the bottom right consists of three parts:

- 1) A list of all conditional branches with one child activity each.
- 2) The condition tree of the currently selected branch.
- 3) The details of the currently selected condition.

The example in Figure 54 uses an empty else-branch. The input image of the workflow is only binarised if certain conditions are fulfilled, otherwise it is not changed. To be able to model such branches, the input data must be routed to the output port of the if-else activity. Currently, that is not possible within the prototype system. Instead, an empty atomic activity is used, which passes through any incoming data.

The next subsection describes how semantic labels can be assigned within the workflow system.

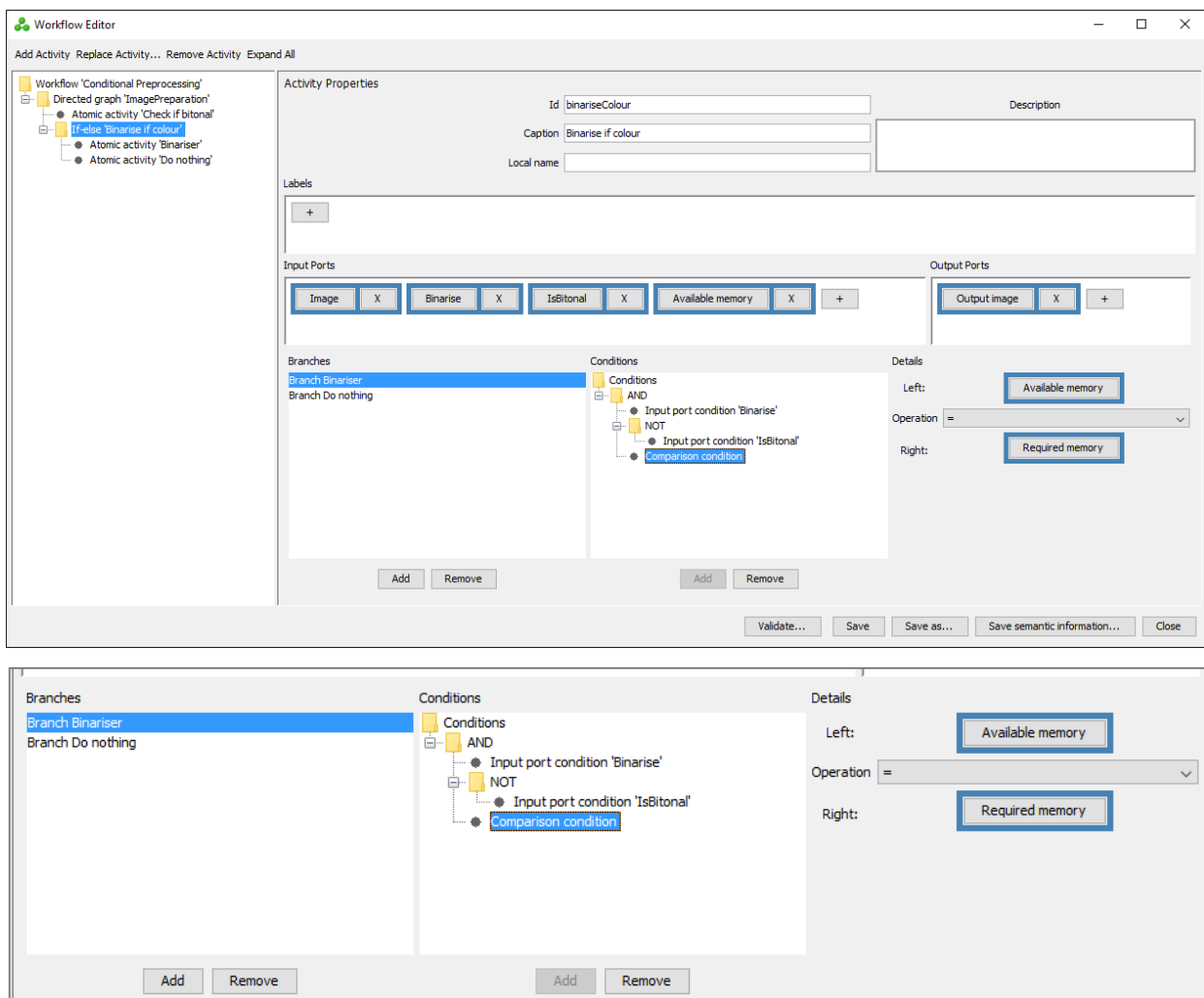


Figure 54 - Details of If-Else activity in Workflow Editor (bottom: branches and conditions enlarged)

### 5.4.3 Semantic Labels

Data ports and activities can be annotated with semantic labels. This functionality can be accessed via dedicated panels in the Data Port dialogue (Figure 47 on page 101) and the activity details section (Figure 50 on page 104).

New labels can be chosen from the ontology within a dedicated dialogue (Figure 55). Once a label of a specific type has been assigned, no second label with the same type can be added. The allowed number of labels from a label group is restricted by the corresponding cardinality (number of label slots) that is defined in the ontology.

Assigned labels are displayed as “tiles” with heading (the root type), type name (button), and a button to remove the label (“X”). Clicking the type name opens a dialog showing the whole branch of the label type in the ontology (from the selected type to the root type).

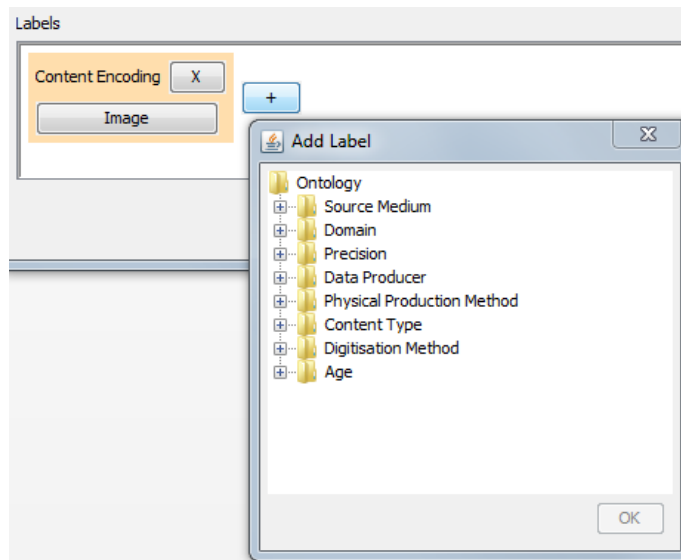


Figure 55 - Dialogue for adding a semantic label (here: label for a data port)

### 5.4.4 Data Tables

Data tables represent a special concept of a workflow-wide data source (see also 4.1.3). Figure 56 shows the diagram for the related interfaces and classes.

A data table represents a two-dimensional matrix of data objects. Data tables can be used as data sources within a workflow or as part of a data repository (see 5.3.2). Unlike single

data objects or data collections, they are not part of the dataflow of a workflow (i.e. they are not passed around between activities).

A workflow can have multiple data tables, all of which can be accessed from each activity of the workflow. The columns of a table resemble output ports of an activity, in fact, they are implemented using the same Java interface within the prototype implementation. Figure 57 shows the dialogue for selecting the source of a data port. All tables and their columns are listed as viable source.

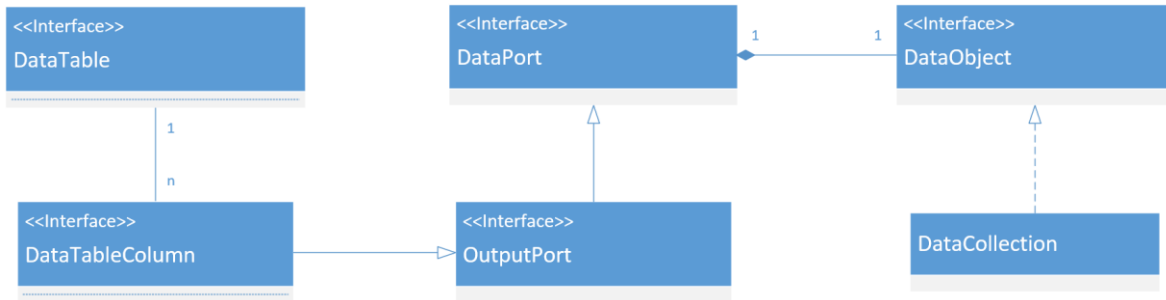


Figure 56 - UML class diagram for data tables

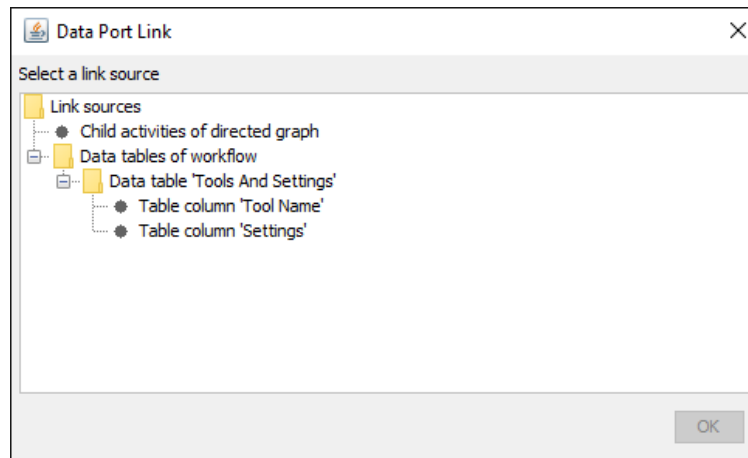


Figure 57 - Table columns as data source

Data tables can be created in the Workflow Editor and they appear as first child nodes under the workflow root in the main tree (see Figure 58). ID, caption and description can be

specified in the details panel on the right. Table columns are added similarly to activity input or output ports and can be annotated with semantic labels.

A table cell is represented by a “DataObject”. Although data objects can also be collections, cells are currently limited to single data objects. The content of a table can be edited with a dedicated editor (Figure 59). Adding values fills the “Fixed Content” attribute that is also used for input ports of activities.

The next subsection discusses the implementation of abstract workflows (templates).

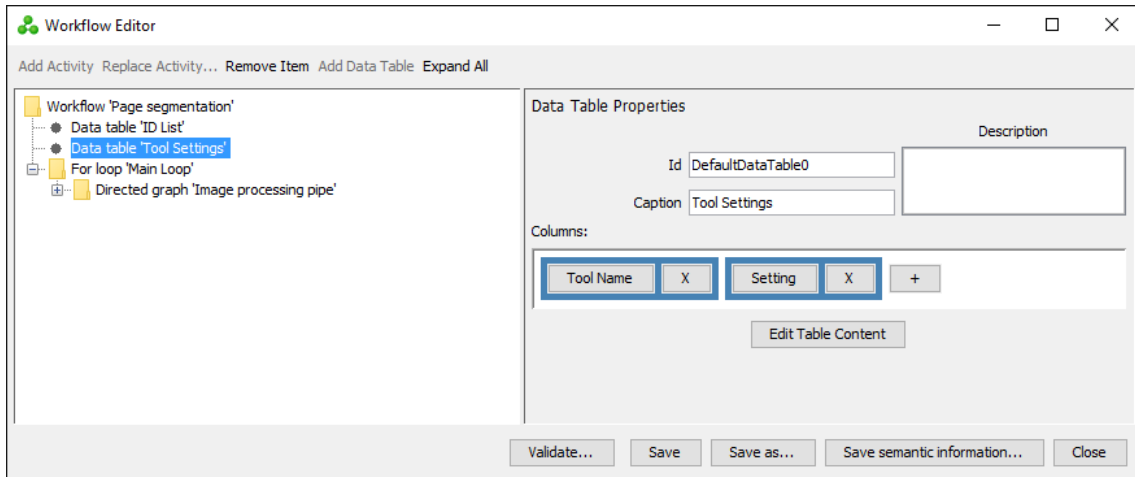


Figure 58 - Data tables in the Workflow Editor

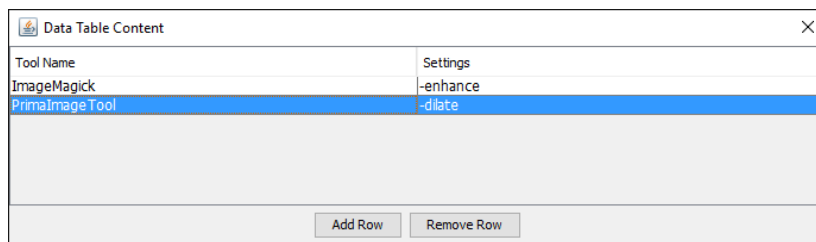


Figure 59 - Editing the content of data tables

### 5.4.5 Workflow Templates

The concept of templates (see Section 4.1.4) is realised by means of abstract workflows. A workflow is abstract if it contains one or more abstract activities.

The workflow editor shows the status of abstractness in the workflow properties (Figure 60) and activity properties. Control flow activities are implicitly abstract if they have an abstract child activity. Atomic activities are explicitly marked as abstract or not abstract.

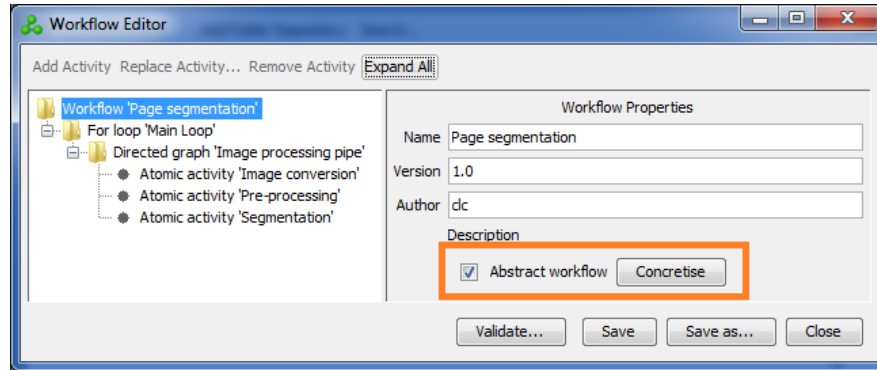


Figure 60 - Workflow properties

The editor contains a method for *concretisation* of the current workflow, wherein abstract activities are automatically (or with user interaction) replaced by matching concrete activities from a specified repository (more details in subsection 5.4.8).

The algorithm for semantic activity matching (used during concretisation) is described in the next subsection.

### 5.4.6 Activity Matching

An activity matching algorithm (see also Section 4.3.1) was developed that is based on the following properties: activity labels, data port labels, and data types. The matching is carried out by comparing a set of reference properties against multiple candidate sets of properties. Depending on the task, the reference properties have different origins. When replacing an activity, the properties are derived from the activity that is to be replaced (input/output ports and activity labels). On the other hand, when adding a nested activity, the properties are aggregated from the parent activity and possibly siblings (if the target is a directed graph activity).



Figure 61 shows an overview of the classes and interfaces used for matching. A more complete class diagram can be found in Appendix B (page 244). Central are the interfaces *Matcher* and *MatchValue*. Based on labels and/or data types as inputs, a matcher calculates a match value. The match value contains a score (high means good match) and match details.

The general approach for matching data types or labels is the same. Each reference property is compared against all properties of the current target set. The match score (0 to 100%) is calculated as a composite of all individual comparisons. Eventually, one match score for each pair of reference and target property set is obtained. The details of each score (e.g. compared properties and reason for low value) are thereby collected for possible inspection by the user. Listing 2 shows a simplified version of the matching algorithm for one object with semantic labels.

A special case is the combination of semantic label-based matches and data-type-based matches at activity level. If the reference activity does not specify a data type, the data-type-based match score will always be 100% (because any type is allowed). However, this score carries little information since there was nothing to match. To combine this score with the label-based score with equal weight (arithmetic mean) reduces the impact of the other score which carries more information (semantic match). As a counter-measure, a weighted arithmetic mean is used instead. The weight is higher, if the matching carries more information (i.e. if the reference activity specifies a data type).

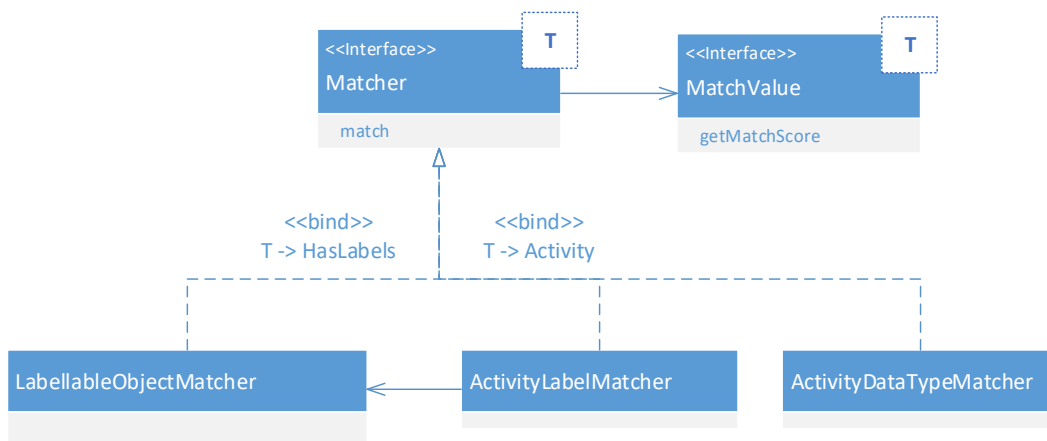


Figure 61 - UML class diagram for matchers

Listing 2 – Label-based matching for one object (pseudocode)

```
Inputs:
  referenceObject (object with semantic labels)
  objectToMatch (object with semantic labels)

//Calculate average of match scores of all labels of the reference object
scoreSum = 0
matchCount = 0
for each label of referenceObject (referenceLabel)
  //Find best match between referenceLabel and any label of objectToMatch
  maxScore = 0
  for each label of objectToMatch (labelToMatch)
    score = matchLabel(referenceLabel, labelToMatch)
    if (score > maxScore)
      maxScore = score
  scoreSum += maxScore
  matchCount++
return scoreSum / matchCount (average score)

function matchLabel(referenceLabel, labelToMatch)
  //Root label type? -> No match (root types are just general categories)
  if (referenceLabel.parent == null)
    return 0%
  //Equals? -> Full match
  if (referenceLabel == labelToMatch)
    return 100%
  //Different root type? -> No match
  if (referenceLabel.root != labelToMatch.root)
    return 0%
  //Reference label subtype of label to match? -> No match
  //(reference label is more generic than the label to match)
  if (referenceLabel.isSubtypeOf(labelToMatch))
    return 0%
  //Partial match -> recursion (lowers score by 10%* each time)
  //(reference label is specialised child of label to match)
  return matchLabel(referenceLabel.parent, labelToMatch) - 10%
```

(\*) The value of 10% was chosen heuristically. The exact value does not matter since match scores are used for relative comparison between several matches. What is important is that a partial match has a lower score than a full match.

Match results are presented in order of score. Figure 62 shows the workflow editor's dialogue for replacing an existing activity. The user can select a repository and apply search filters if desired. In addition, certain aspects of the matching can be specified (to match by labels only, for instance).

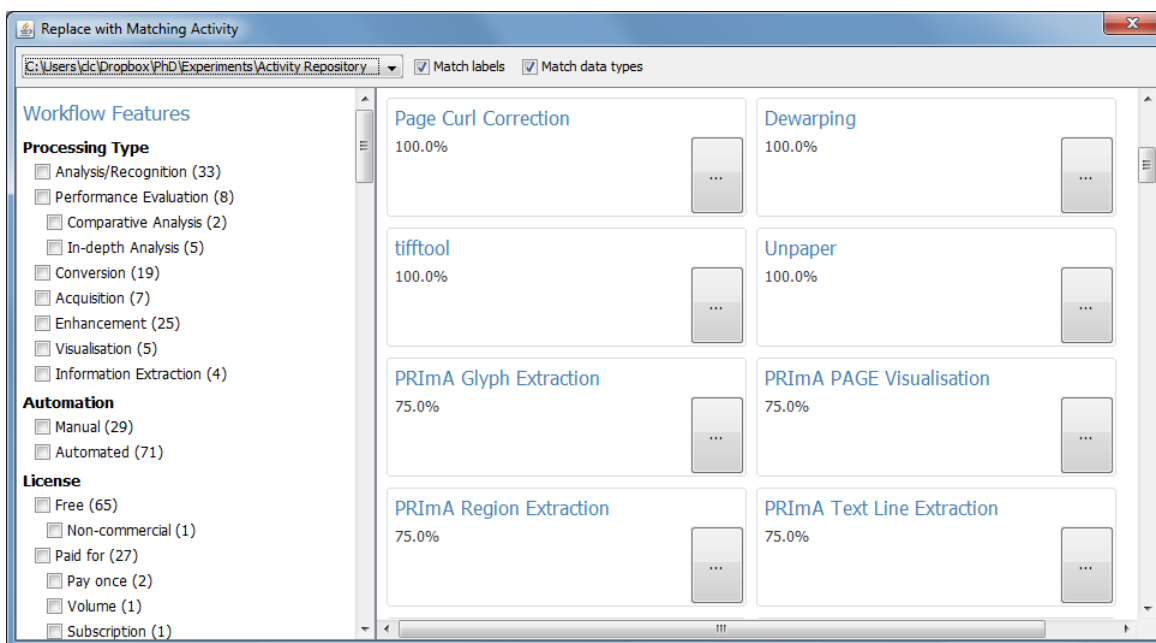


Figure 62 - Matching for replacing an existing activity

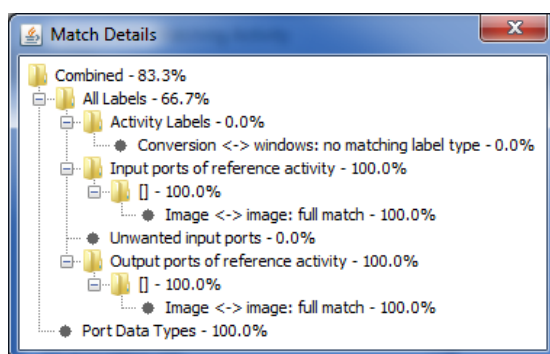


Figure 63 - Details of match result

Details on how the match score was composed can be retrieved on demand (Figure 63). The actual replacement can be initiated by selecting a result item. In a final step, the data ports of the existing and the new activity must be aligned, and metadata can be confirmed or modified (see Figure 64).

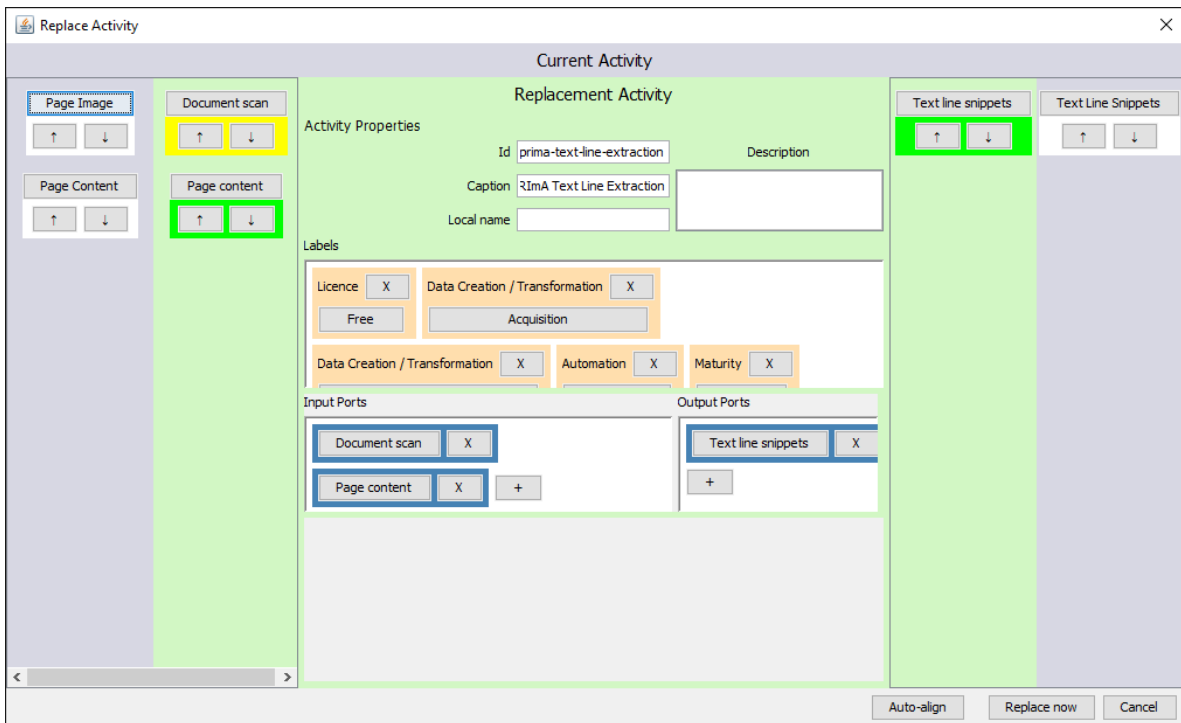


Figure 64 - Port alignment and metadata specification for replacing an activity

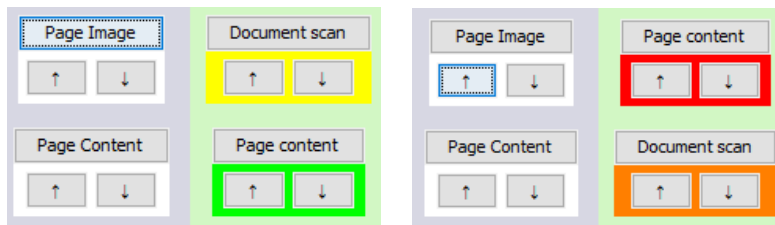


Figure 65 - Port alignment: Left: well aligned; Right: misaligned

The matching for adding a new child activity works in a similar fashion, with the exception that the port alignment can be omitted.

The dialogue for replacing an activity visualises both old and replacement activity at once in a nested way. The current activity (coloured in purple) and its data ports are wrapped around the replacement activity (green). Input and output ports of the two activities are shown side-by-side to enable a quick visual association. The user can change the position of each data port by using up or down arrows. Each data port of the replacement activity is also colour-coded according to how well it fits the counterpart of the old activity (see Figure 65).

The colour thereby ranges from green (perfect match) to red (no match). The match scores are calculated using the same algorithm that is also used for finding a matching activity.

Using the port matching approach, an *auto-alignment* feature was implemented. The algorithm finds and assigns the best matching port for each reference port. Non-matching ports are added at a position after the last reference port and therefore have no counterpart.

The next subsection introduces *validation* – a feature to help to create complete and usable workflows.

### 5.4.7 Workflow Validation

Validation is an assistive feature to improve the quality of workflows by analysing certain aspects and providing the user with feedback (see also Section 4.3.2). Validation was added to the prototype system using a modular design that promotes extensibility (see Figure 66).

The following checks are available: workflow object validation, activity validation, missing child activities, cycle detection, unconnected data ports, data type matching, data cardinality matching, missing data types, and semantic label matching.

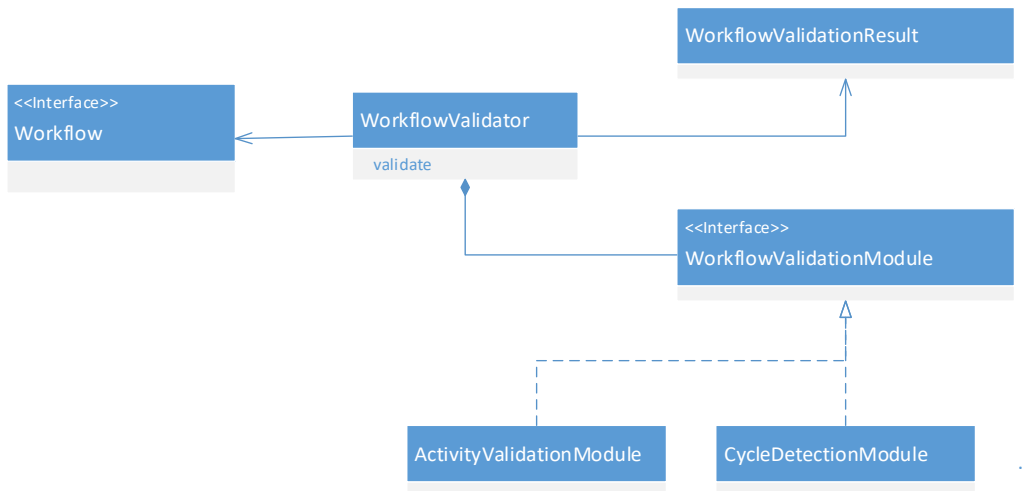


Figure 66 - UML class diagram for workflow validation

Validation results are presented hierarchically with three main categories: errors, warnings, and notes (“info”). Each result item contains detailed information on the nature of the corresponding issue. Where applicable, items are also linked to a corresponding workflow component (e.g. an activity). Selecting the result item then prompts the main workflow editor to show the linked component.

Figure 67 shows the validation dialogue with example results. A specialised panel at the bottom of the window offers controls to directly solve the corresponding problem or a step-by-step description for a manual resolution.

The activity matching and validation represent semi-automated functionality. The next section describes the workflow composition features that do not require any user interaction.

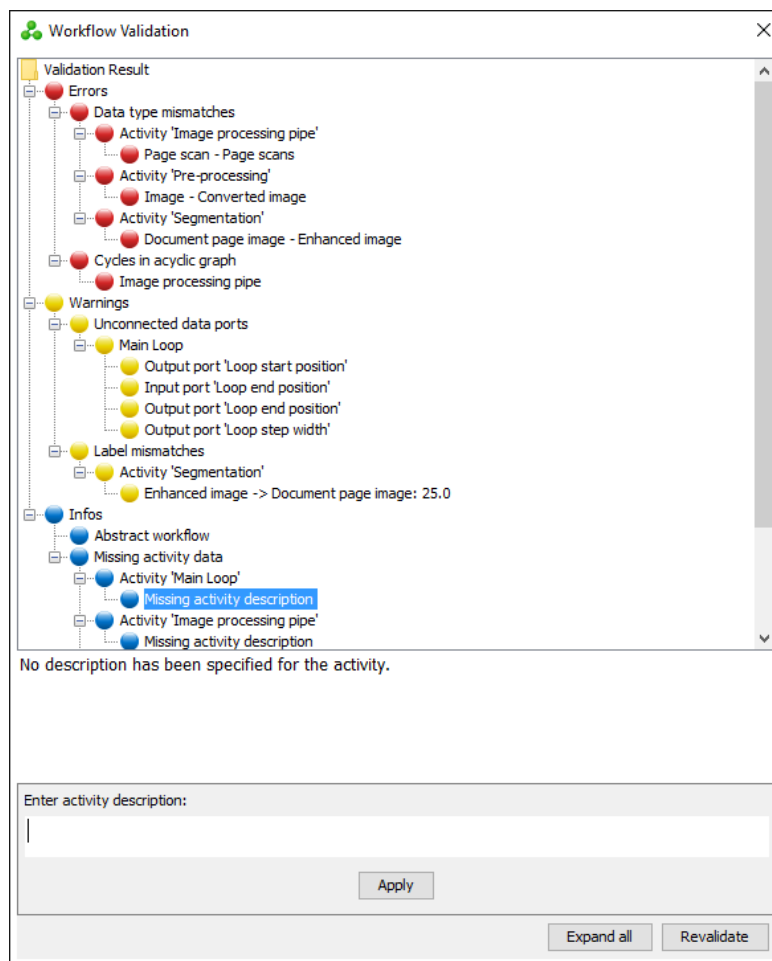


Figure 67 - Validation dialogue

### 5.4.8 Concretisation and Data Conversion

*Workflow concretisation* was outlined in Section 4.3.3 as means of making a workflow template executable by substituting placeholders with non-abstract activities from a provided repository (activity repositories are realised using workflow repositories that only contain workflows with exactly one atomic activity).

The developed algorithm for the workflow concretisation makes use of the activity matching described in Section 5.4.6. Listing 3 provides an overview.

*Listing 3 – Workflow concretisation algorithm (pseudocode)*

```
function concretise(Workflow workflow, Activity[] activityRepository, boolean interactive) {
    Activity[] abstractActivities = workflow.findAbstractActivities();

    sortAscendingByNumberOfBestMatches(abstractActivities, activityRepository);

    for each activity in abstractActivities {
        Activity[] concreteMatchingActivities = findBestMatches(activity, activityRepository);
        if (concreteMatchingActivities.size > 1 && interactive)
            refineMatchingResultByUser(concreteMatchingActivities);
        workflow.replaceActivity(activity, concreteMatchingActivities[0]);
    }
}
```

Match scores are calculated similarly to the method described earlier, taking into account semantic labels and data types of data input and output ports as well as labels of the activity itself. One difference is made (when running the automated concretisation) with how data type mismatches are handled. Contrary to the interactive tasks, a mismatch is treated in a stricter manner, essentially excluding all activities from the repository which do not fit the data type profile (or type compatibility).

A “Strictness” threshold (adjustable by the user, see Figure 68 top right) is used to reject activities that have a low match score. The lower the threshold, the higher is the chance of concretising the whole workflow. However, using a low threshold risks producing a workflow that is not fit for purpose.

The order in which abstract activities are replaced by concrete ones is defined by the number of best matching activities (all activities from the repository with the highest match score, as long as the score is not zero). The purpose of this sorting is to replace the activities with the least number of matches first. The information concerning which activities have been replaced can help in choosing the following activities (which have more options for replacements).

The replacement of a single activity is carried out using the automated data port alignment that was discussed in Section 5.4.6. If an abstract activity is replaced by another abstract activity (e.g. a template), the concretisation process is repeated recursively.

At the end of the process the concretisation dialogue shows an overview of the replacements that have been carried out and a success indicator. Figure 68 shows the concretisation result for the template that is introduced in Figure 60 (page 111). Examples for interactive workflow concretisation are described in Chapter 6.

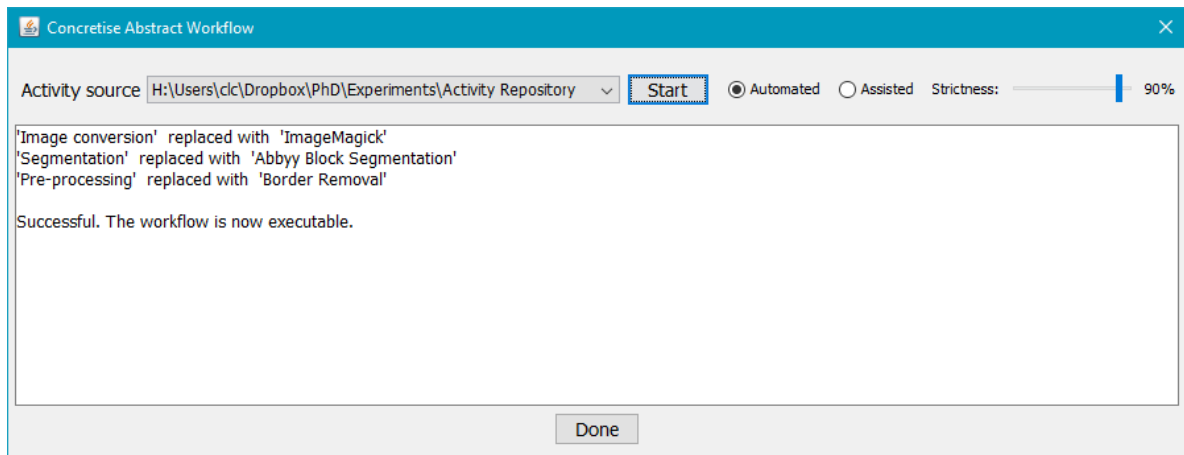


Figure 68 – Workflow concretisation

Automated data conversion was introduced in Section 4.3.3. It represents a way of dealing with data type mismatches (which would lead to an error at execution time).

The prototype reports data type mismatches during workflow validation. A specialised user interface panel offers an automated and a manual way of correcting the problem (see Figure 69).



The automated approach adds a conversion activity and re-routes the dataflow through the new activity. The algorithm can be summarised as follows:

For a given source activity with a source data port and target activity with a target data port:

- (1) If the parent activity of the target activity is not a directed graph activity, replace the parent activity with a new graph activity and add the target activity as a child
- (2) Add an abstract atomic converter activity to the directed graph activity
- (3) Add an input and an output port to the converter activity, using the data types from the source port and the target port
- (4) Link the source port to the input port of the converter activity
- (5) Link the target port to the output port of the converter activity
- (6) Invoke the concretisation procedure (see above)

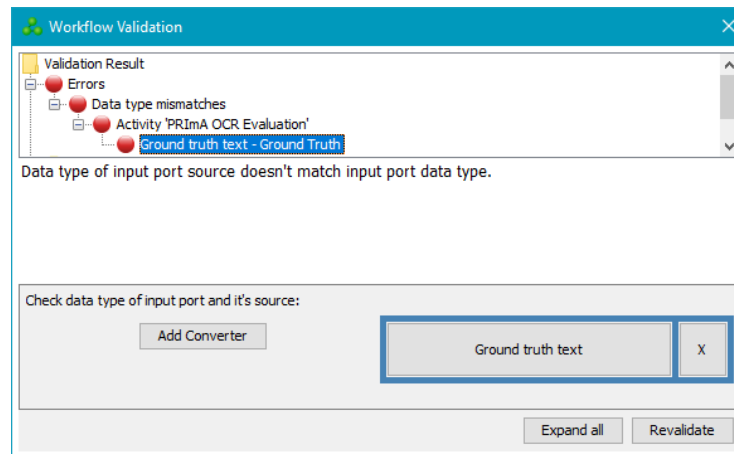


Figure 69 - Data type mismatch workflow validation error

#### 5.4.9 Workflow XML Format

Workflows are stored using a special XML-based format. Listing 4 shows a shortened version of the example workflow in Figure 53 (page 106). Each activity type has a corresponding XML element. The concept of child activities can be translated directly to nested XML elements. The example has a for-loop activity containing a DAG activity which in turn contains an atomic activity.

Listing 4 - Example workflow in XML format (simplified)

```
<Workflow author="clc" name="Page segmentation" version="1.0" ontologyVersion="2">
<Description>...</Description>
<DataTable ...>
  <TableColumn ...>
    ...
  </TableColumn>
</DataTable>
<ForLoopActivity caption="Main Loop" id="main_loop">
  <InputPort allowedTypes="file.image" id="InputPortImpl0">
    <DataCollection caption="Page scans">
      <Label type="domain.dia"/>
      ...
    </DataCollection>
  </InputPort>
  <OutputPort id="OutputPortImpl0" positionSource="LoopPortImpl7" ...>
    <DataCollection caption="Page layout files"/>
  </OutputPort>
  ...
  <DirectedGraphActivity caption="Image processing pipe" id="processing-pipe">
    <InputPort allowedTypes="" id="InputPortImpl2" ...>
      <SingleDataObject caption="Page scan"/>
    </InputPort>
    <OutputPort id="OutputPortImpl2" source="OutputPortImpl1" type="">
      <SingleDataObject caption="Page segmentation file"/>
    </OutputPort>
    ...
    <Vertex height="30" width="128" x="65" y="130" ...>
      <AtomicActivity abstract="true" caption="Image conversion" ...>
        ...
      </AtomicActivity>
    </Vertex>
    <Vertex height="30" width="128" x="265" y="130" ...>
      <IfElseActivity caption="Selective Image Enhancement" ...>
        <Branch ...>
          ...
        </Branch>
      </IfElseActivity>
    </Vertex>
  </DirectedGraphActivity>
</ForLoopActivity>
</Workflow>
```

Semantic labels are stored as sub-elements within the workflow components that can be labelled (activity, data object). As mentioned in subsection 5.2.3, major changes in the used ontology require some form of migration solution. A prerequisite for such a process is to include information (in the workflow) on which ontology was used to create the labels. The Ontology Editor (see 5.2.1) allows the definition of an ontology version number (an Integer). When saving a workflow, it is annotated using the current ontology version and, in turn, when

opening a workflow, its labels can be migrated to the latest revision. The migration is discussed in the next subsection.

### 5.4.10 Ontology Migration

When a workflow is loaded from a file, any contained semantic label must be part of the current ontology, otherwise it will be ignored (the system currently allows a single ontology). Nevertheless, an ontology may contain migration rules to translate labels of an older ontology to the new version (see again 5.2.3). Figure 70 shows this process in context of parsing a workflow XML file. The algorithm ensures that the loaded workflow only contains valid labels. Saving the updated workflow therefore means invalid labels will be lost.

The Workflow Editor notifies the user in case labels have been changed or ignored (see Figure 71).

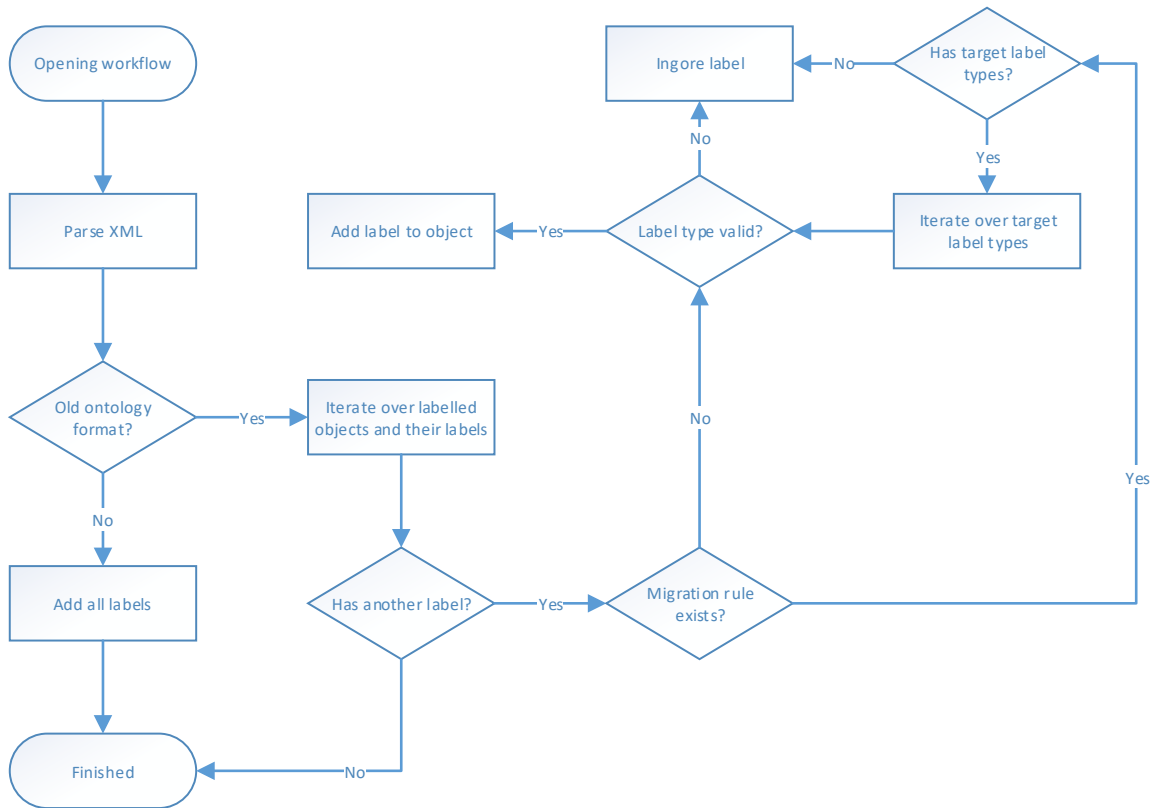


Figure 70 - Flow chart for label type migration

This concludes the description of the basic implementation of the workflow system including: ontology editor, repositories, and workflow editor. Before discussing support for the Web Ontology Language, a short final subsection introduces a visualisation approach for workflows.

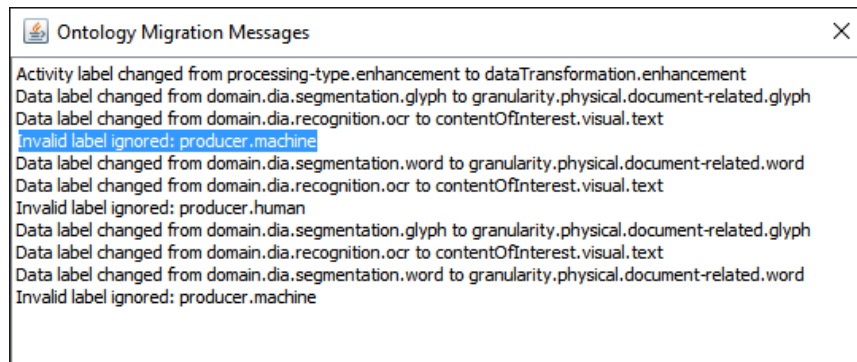


Figure 71 - Ontology / label migration messages

#### 5.4.11 Workflow UML Visualisation

An experimental workflow renderer was added to the Workflow Editor that uses a UML-like activity graph. For simplicity, directed graph activities are drawn in a horizontal layout (child activities one after the other) and if-else activities are drawn in a vertical layout (branches stacked one over the other).

Figure 72 shows an example with a graph activity, Figure 73 shows an example with if-else activity, Figure 74 shows a for-loop activity (dashed), and Figure 75 shows a more complex workflow with all activity types.

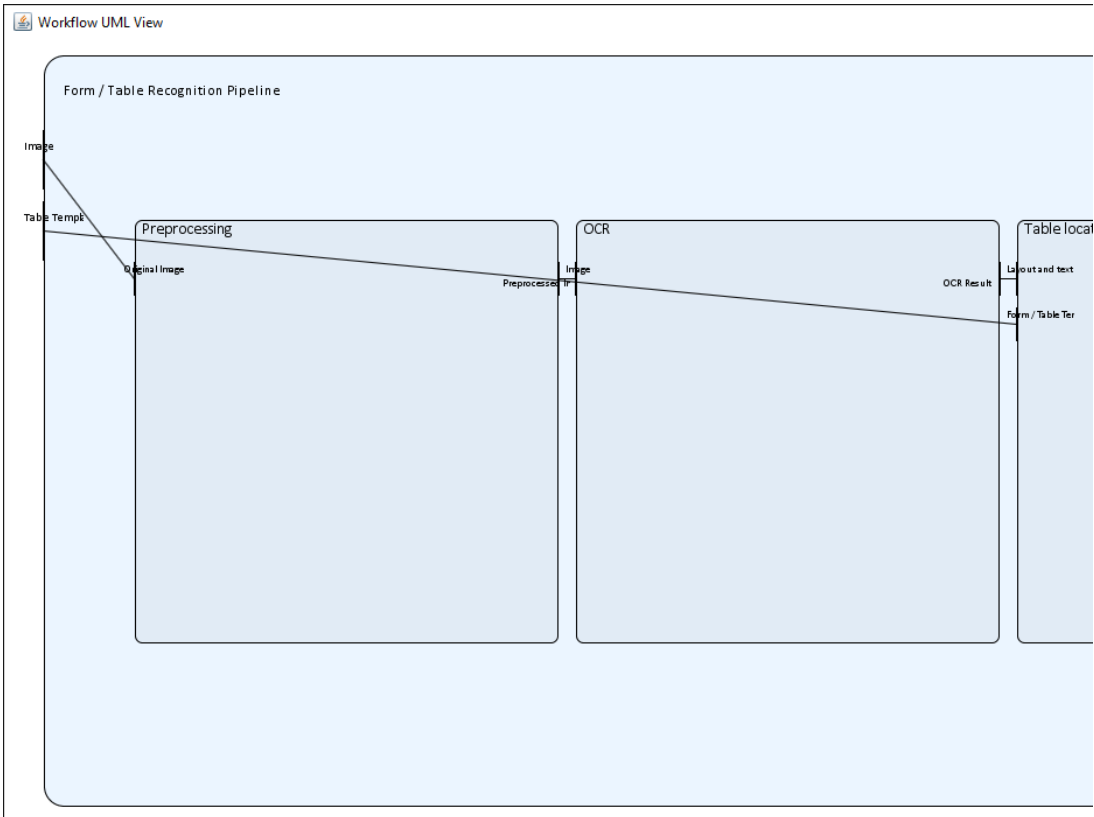
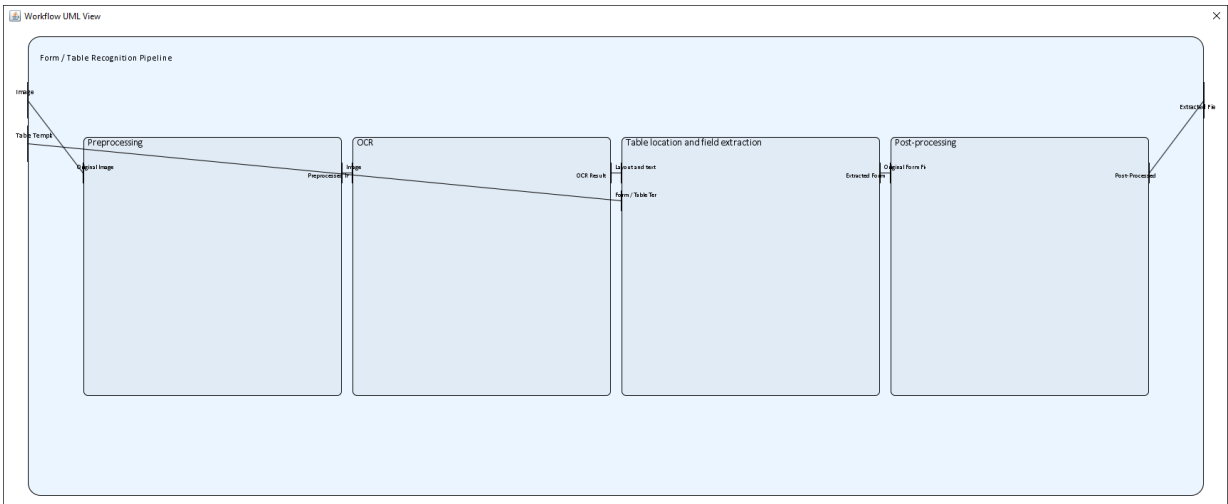


Figure 72 - UML-like workflow visualisation of example with directed graph activity (top: entire workflow; bottom: left part enlarged)

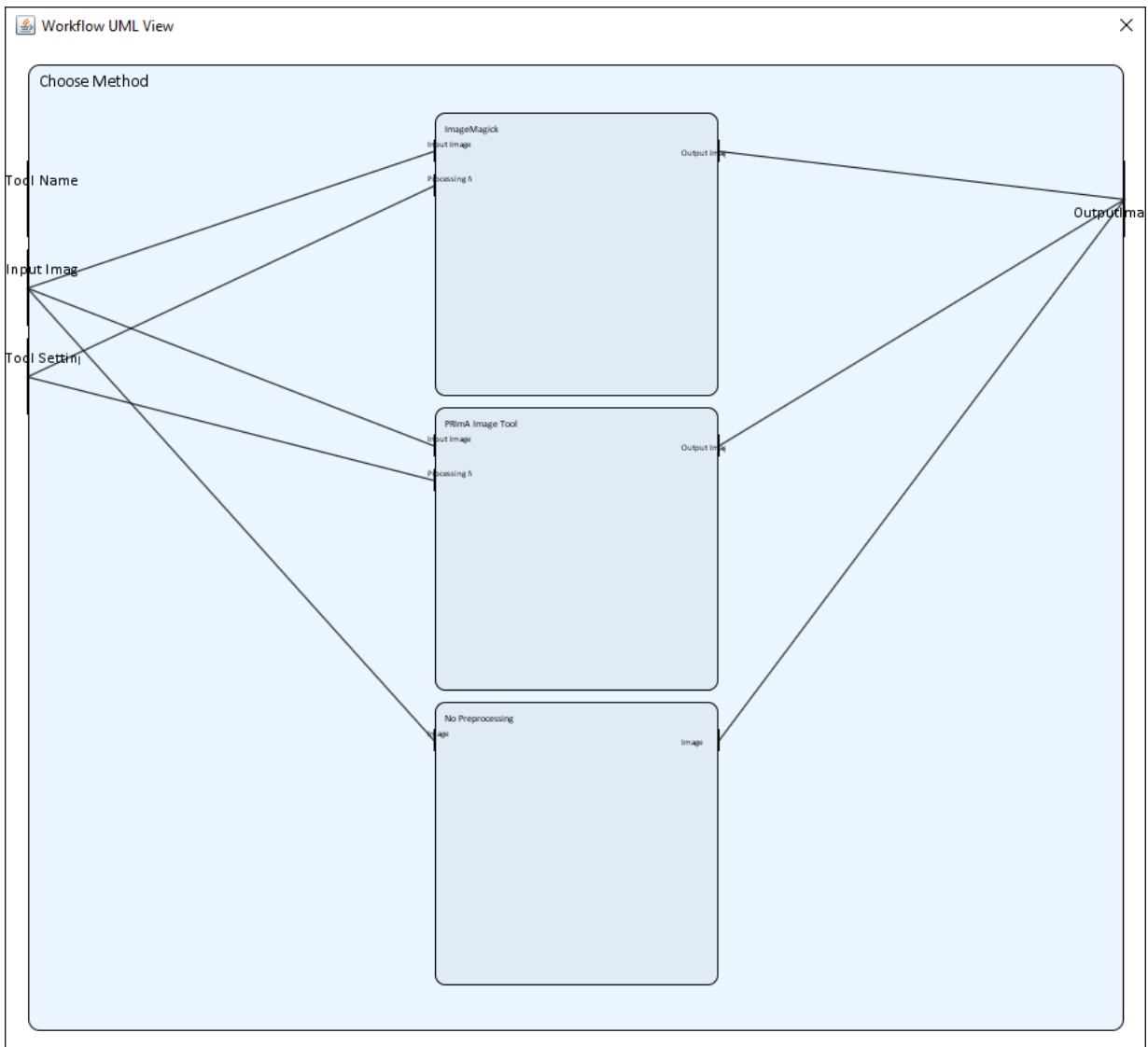


Figure 73 - UML-like workflow visualisation of example with if-else activity

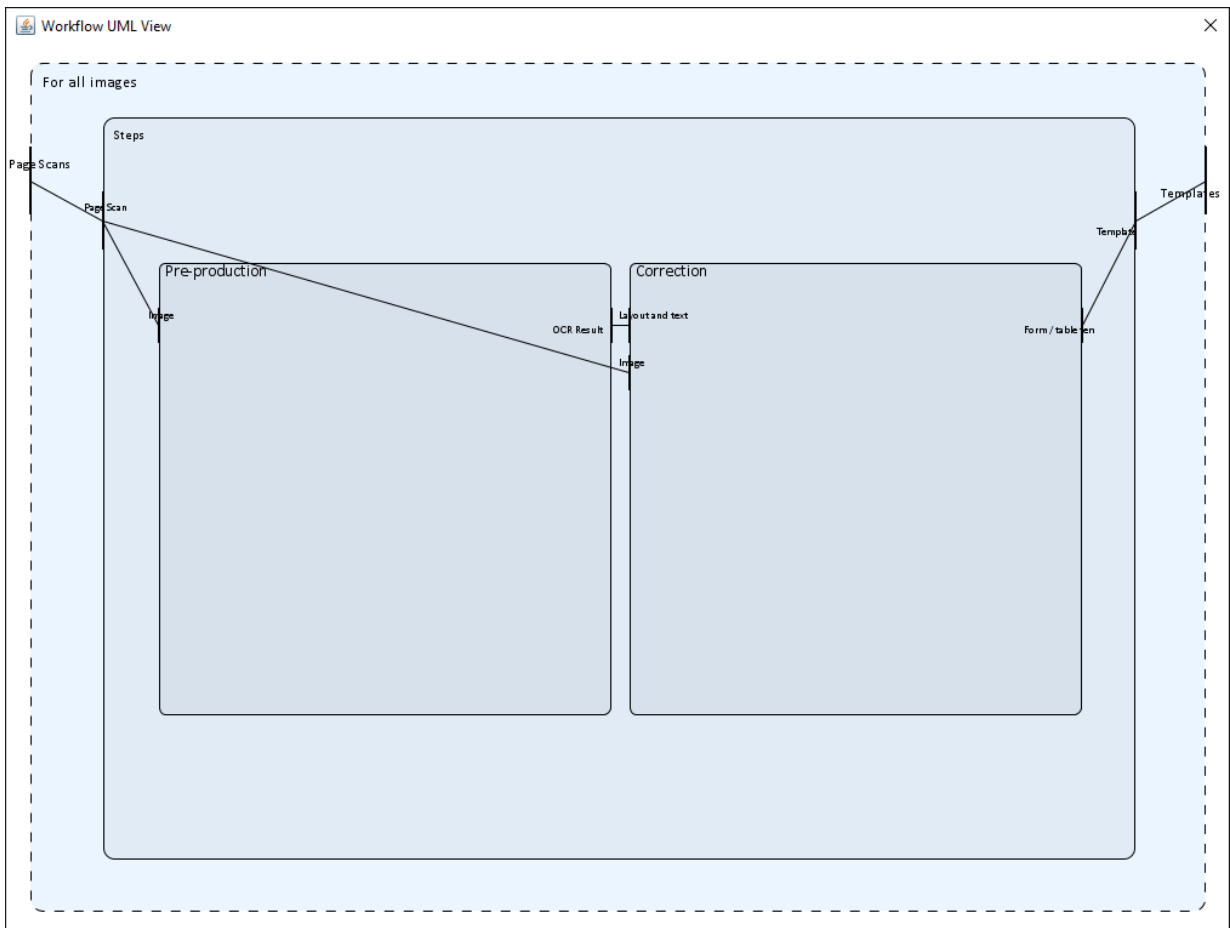


Figure 74 - UML-like workflow visualisation of example with for-loop activity

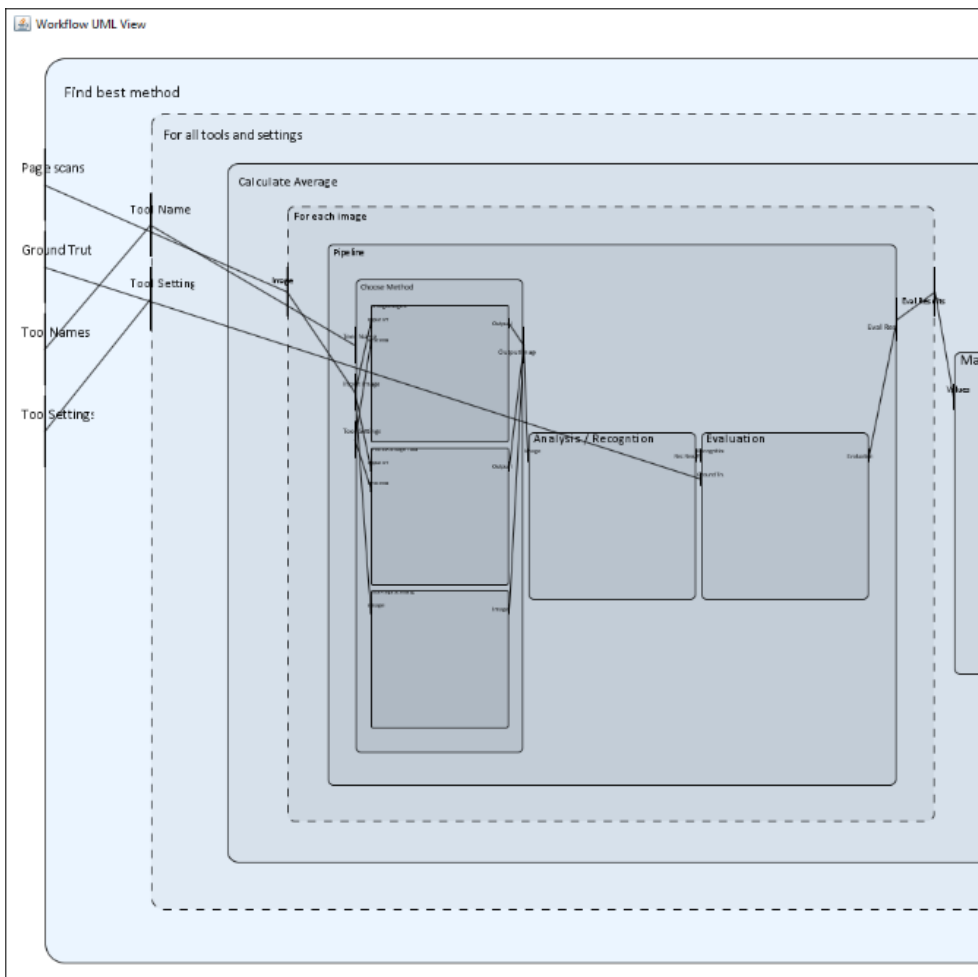
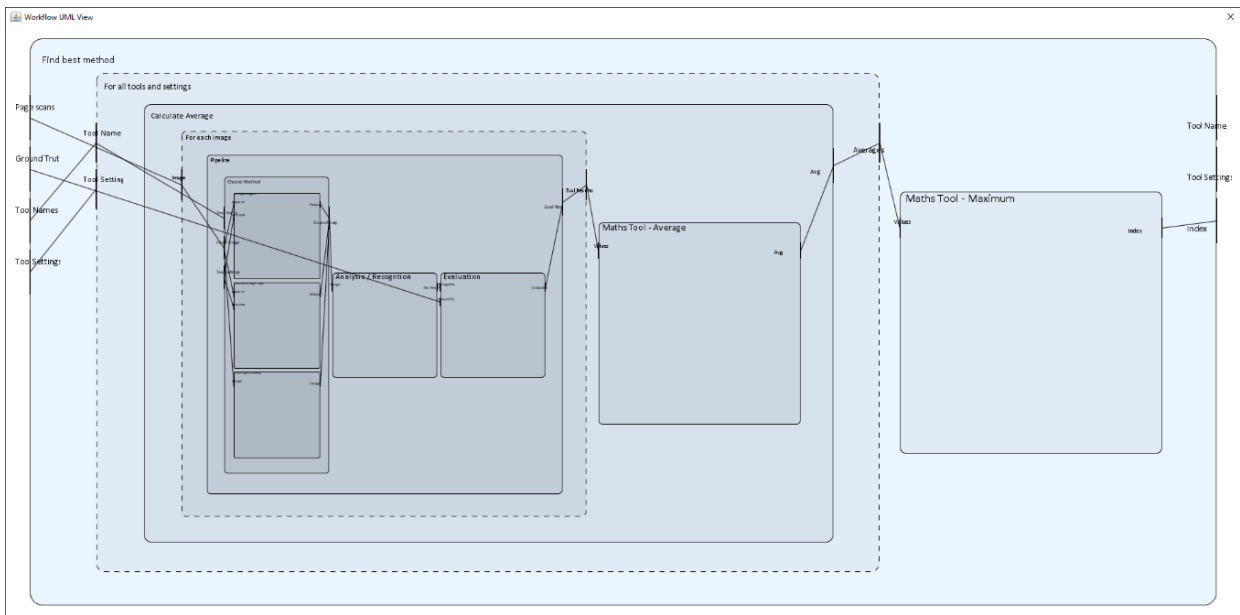


Figure 75 - UML-like workflow visualisation of complex example (bottom: enlarged left part)



## 5.5 Web Ontology Language Support

The label-centric ontology and the semantic information of a workflow are a subset of the Web Ontology Language (OWL) in terms of expressiveness. To prove this claim, the prototype has been extended by an export module that saves the semantic data in OWL2 XML format [4]. A standardised format also makes the data interchangeable with other systems and enables the use of third-party semantic reasoners, for example.

OWL is widely used in the context of the Semantic Web and the respective OWL2 XML format is the most recent recommendation of the World Wide Web Consortium (W3C).

The native data formats of the prototype are two specially developed XML formats for storing ontologies and workflows (including semantic information). The aim was therefore to define a mapping from the data model of the system to OWL that leads to equivalent persistently stored data. The decision to use proprietary formats was made in the early stages of the PhD research, in expectation of benefits such as: faster implementation, better readability (by a human), and more efficiency in terms of file size. This is discussed at the end of this section.

### 5.5.1 Label Types in OWL

As a first step, the functionality to save an ontology containing a label type hierarchy was developed. As described in Section 5.2, this includes label taxonomies and label slots. Both can be described using OWL classes and object properties. Table 4 shows the mapping for all aspects of a label type ontology.

*Table 4 - Mapping of XML format aspects for ontologies (dedicated vs OWL2)*

Ontology aspect	Implementation concept and example for dedicated XML format	Implementation concept and example for OWL2 XML format
Label type declaration	Dedicated XML element <LabelType name="..."/>	Class declaration <Declaration> <Class IRI <sup>4</sup> ="#..."/> </Declaration>

---

<sup>4</sup> IRI: International Resource Identifier

Ontology aspect	Implementation concept and example for dedicated XML format	Implementation concept and example for OWL2 XML format
Label type hierarchic relation	Nested XML elements <pre>&lt;LabelType name="..."&gt; parent   &lt;LabelType name="..."&gt; child &lt;/LabelType&gt;</pre>	Explicit class relationship definition <pre>&lt;SubClassOf&gt;   &lt;Class IRI="#..."&gt; child   &lt;Class IRI="#..."&gt; parent &lt;/SubClassOf&gt;</pre>
Label type metadata (e.g. caption and description)	XML attributes and text elements <pre>&lt;LabelType caption="..." ...&gt; &lt;Description&gt;...&lt;/Description&gt; &lt;/LabelType&gt;</pre>	OWL annotation properties (declaration and assertion) <pre>&lt;Declaration&gt;   &lt;AnnotationProperty IRI="#labelCaption"/&gt; &lt;/Declaration&gt; &lt;Declaration&gt;   &lt;AnnotationProperty IRI="#labelDescription"/&gt; &lt;/Declaration&gt; &lt;AnnotationAssertion&gt;   &lt;AnnotationProperty IRI="#labelCaption"/&gt;   &lt;IRI&gt;#...&lt;/IRI&gt; class   &lt;Literal datatypeIRI = "&amp;rdf;PlainLiteral"&gt;...&lt;/Literal&gt; &lt;/AnnotationAssertion&gt; metadata</pre>
Allowed label assignments	Explicit XML elements with nested elements <pre>&lt;ActivityLabelSlots&gt;   &lt;LabelSlotGroup name="..."   .../&gt; &lt;/ActivityLabelSlots&gt; &lt;DataObjectLabelSlots&gt;   ... &lt;/DataObjectLabelSlots&gt;</pre>	Classes and object properties <pre>&lt;Declaration&gt;   &lt;Class IRI="#Activity"/&gt; &lt;/Declaration&gt; &lt;Declaration&gt;   &lt;Class IRI="#DataObject"/&gt; &lt;/Declaration&gt; &lt;Declaration&gt;   &lt;ObjectProperty IRI="#has..."/&gt; &lt;/Declaration&gt; &lt;ObjectPropertyDomain&gt;   &lt;ObjectProperty IRI="#has..."/&gt;   &lt;Class IRI="#Activity"/&gt; &lt;/ObjectPropertyDomain&gt; &lt;ObjectPropertyRange&gt;   &lt;ObjectProperty IRI="#has..."/&gt;   &lt;ObjectProperty IRI="#has..."/&gt;   &lt;Class IRI="#..."&gt; &lt;/ObjectPropertyRange&gt;</pre>
Label slot cardinality	XML attribute <pre>&lt;LabelSlotGroup ... slots="3"/&gt;</pre>	Cardinality definition <pre>&lt;ObjectPropertyRange&gt;   &lt;ObjectProperty IRI="#has..."/&gt;   &lt;ObjectMaxCardinality cardinality="3"&gt;   &lt;ObjectProperty IRI="#has..."/&gt;   &lt;Class IRI="#..."&gt;   &lt;/ObjectMaxCardinality&gt; &lt;/ObjectPropertyRange&gt;</pre>

As shown, a label ontology can be fully represented by OWL using the corresponding XML-based standard format. A switch within the Ontology Editor allows the user to choose to save the current ontology in the dedicated format or in OWL2 XML format. To ensure syntactical correctness, the prototype validates the produced XML data against the official schema [99].

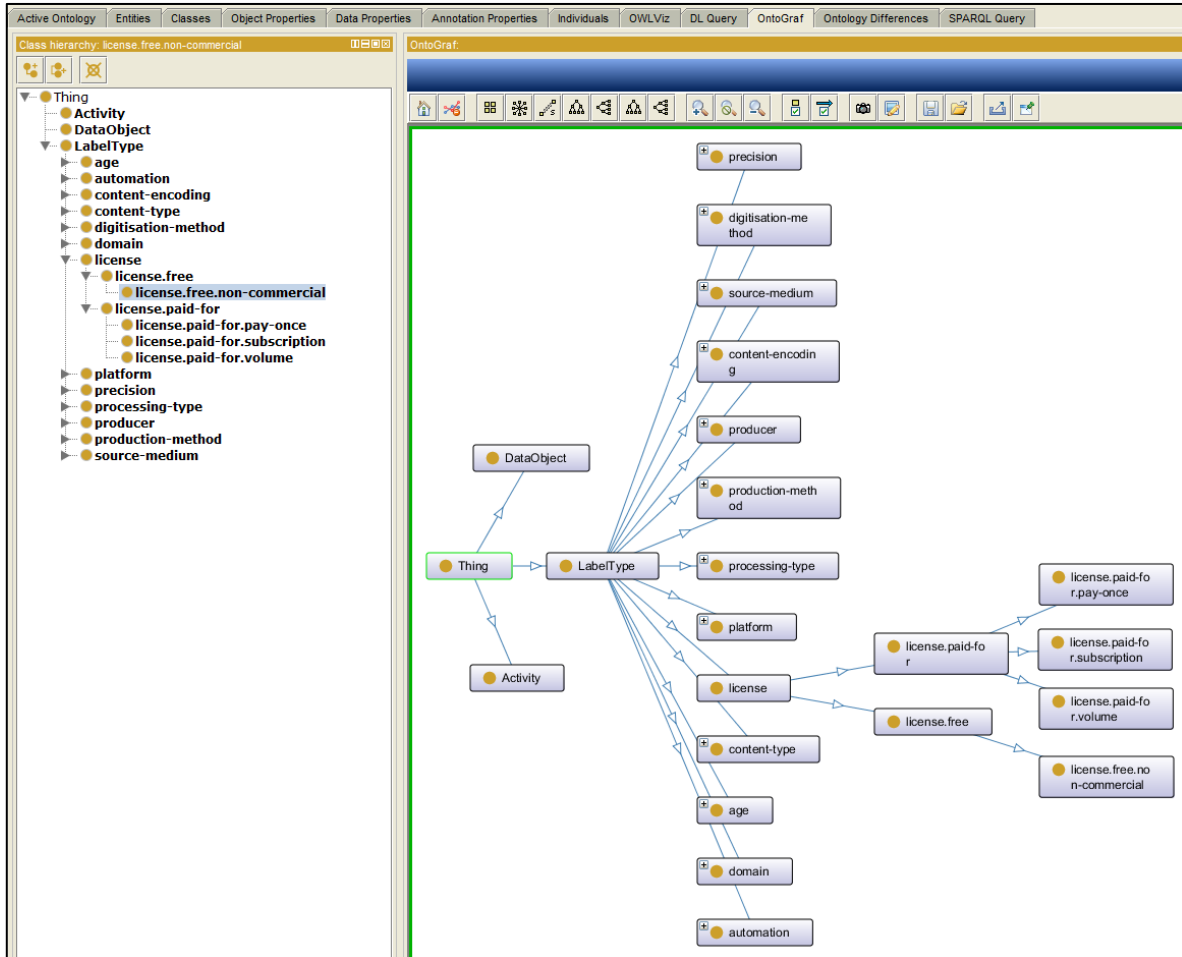


Figure 76 - Label ontology as visualised in Protégé

Further validation was achieved by opening and inspecting a saved ontology with the Protégé Editor (see Figure 76). Two extra classes called “Thing” and “LabelType” can be seen in the screenshot. The former is the super class of all classes in OWL and is created implicitly. The “LabelType” class is not strictly necessary and was added to improve

readability. The fact, that a class is a label type, can also be inferred from the object properties (“#has...”) that link activities and data objects with label slots.

### 5.5.2 Workflow Labels in OWL

While the label type hierarchy represents *terminological* information, the actual labels of the activities and data objects within a workflow represent *factual* (or *assertional*) knowledge. The dedicated XML format of the prototype combines structural and semantic workflow information. The same is not possible with OWL2 XML since OWL is not a workflow representation language. The mapping described in the following therefore applies only for the semantic data within a workflow.

To begin with, in OWL, the factual data about the labels is an extension of the label ontology that is described above. In OWL2 XML this can be expressed through an “<import>” statement, which links to another OWL document via a URL. This requires the label ontology to be available online (technically, a URL can point to a local file but that would drastically limit the usefulness of the ontology).

Table 5 shows the mapping to OWL which is mostly based on *assertions* (class, object property).

Table 5 - Mapping of XML format aspects for semantic workflow data (dedicated vs OWL2)

Ontology aspect	Implementation concept and example for dedicated XML format	Implementation concept and example for OWL2 XML format
Label assignment to activity or data object	Nested XML elements <pre>&lt;Activity&gt;   &lt;Label /&gt; &lt;/Activity&gt;</pre>	Named individuals, class assertions, and object property assertions <pre>&lt;Declaration&gt;   &lt;NamedIndividual IRI="#Activity_...p"/&gt; &lt;/Declaration&gt; &lt;Declaration&gt;   &lt;NamedIndividual IRI="#Activity_..._Label_..."/&gt; &lt;/Declaration&gt;  &lt;ClassAssertion&gt;   &lt;Class IRI="http://...#Activity"/&gt;   &lt;NamedIndividual IRI="#Activity_..."/&gt; &lt;/ClassAssertion&gt;  &lt;ObjectPropertyAssertion&gt;   &lt;ObjectProperty IRI="#has..."/&gt;</pre>

Ontology aspect	Implementation concept and example for dedicated XML format	Implementation concept and example for OWL2 XML format
		<pre>&lt;NamedIndividual IRI="#Activity_..."/&gt; &lt;NamedIndividual IRI="#...Label_..."/&gt; &lt;/ObjectPropertyAssertion&gt;</pre>
Label type	XML attribute <Label type="..." />	Class assertions <pre>&lt;ClassAssertion&gt; &lt;Class IRI="http://...#(label type)"/&gt; &lt;NamedIndividual IRI="#...Label_..."/&gt; &lt;/ClassAssertion&gt;</pre>
Label metadata (e.g. comments)	XML text elements <Label ...> <Comments>...</Comments> </Label>	Data property assertions (not yet implemented)

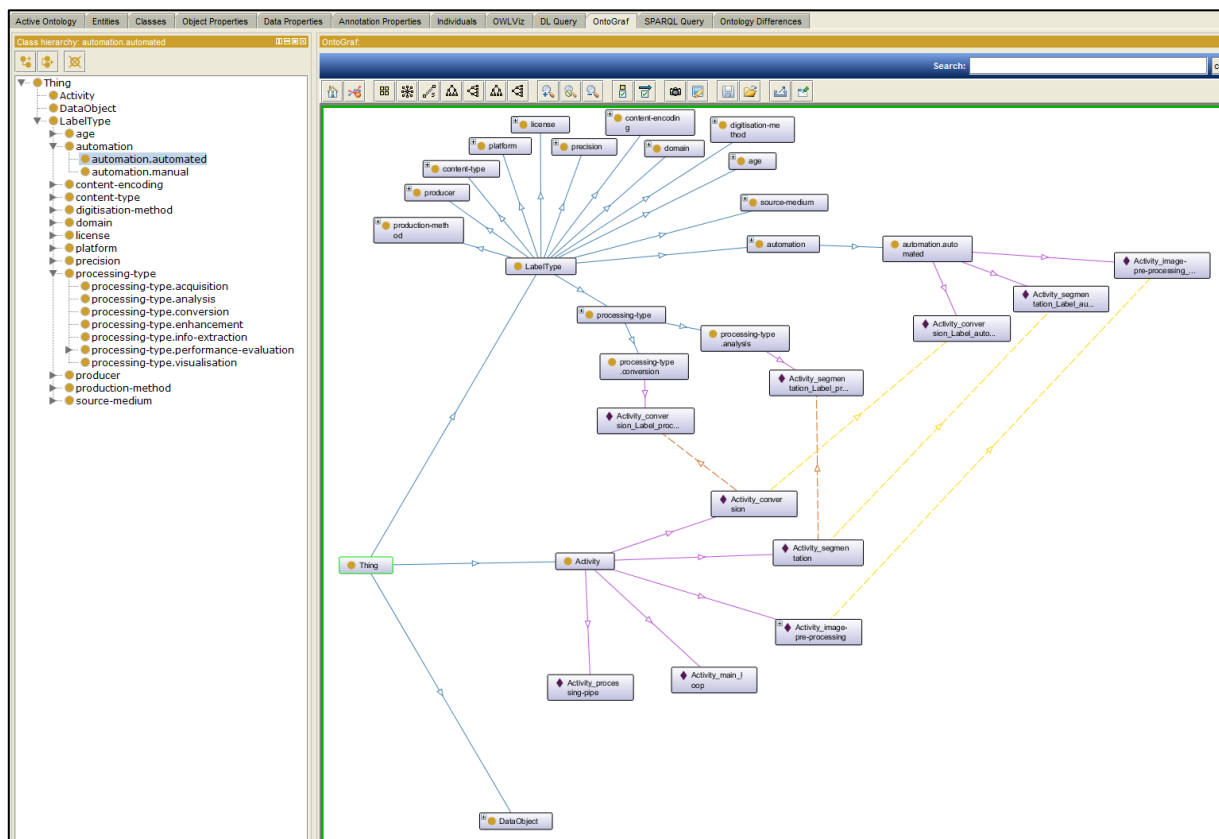


Figure 77 - Semantic workflow data as visualised in Protégé

Within the prototype software, the semantic data can be saved explicitly as OWL2 XML from within the Workflow Editor. Figure 77 shows the semantic data of an example workflow in Protégé. Workflow activities, data objects, and all labels are therein represented by named

individuals. It can be seen that the factual data is indeed an extension of the label ontology – the whole label type hierarchy is included.

### 5.5.3 Discussion and Future Use of OWL

For future iterations of the workflow system it can be considered to fully replace the dedicated label ontology XML format with the OWL2 format. A module to read an ontology from OWL XML has not been implemented yet since it is a straightforward software engineering task and does not provide extra value for this PhD research.

That the proprietary format is indeed more storage space efficient can be predicted from Table 4 and Table 5. A comparison of the file sizes of the example ontology from Section 5.5.2 supports this suggestion. The dedicated XML format leads to a file of 10KB whereas OWL2 XML requires 52KB for the same information.

The readability is arguably better with the specialised XML format. Based on Listing 5 the readers can form their own opinions. Human readability can simplify the development of a system since it helps debugging and testing. Additionally, the dedicated XML format required less implementation effort. One reason was that the concepts of the semantic model and the workflow model could be directly translated to an unconstrained XML format. Another reason is the complexity of OWL, which covers a much wider range of use cases. This was intensified by the limitations of the documentation that is currently available (OWL2 XML is a new format).

The final section of this chapter discusses the complexity and performance of presented algorithms and features.

Listing 5 - Representation of example ontology in proprietary XML and OWL2 XML (shortened)

```

Proprietary XML:
<LabelTypeHierarchies>
  <LabelType caption="Domain" name="domain">
    <Description>A specific step in a processing pipeline ...</Description>
    <LabelType caption="Document Image Analysis" name="dia">
      <LabelType caption="Image Processing" name="image-processing"/>
      <LabelType caption="Image Pre-Processing" name="image-preprocessing"/>
      <LabelType caption="Geometric Correction" name="geometric-correction"/>
      <LabelType caption="Segmentation" name="segmentation">
        <LabelType caption="Region/Block" name="region"/>
        <LabelType caption="Text Line" name="text-line"/>
    ...
  ...

OWL2 XML:
<Declaration>
  <AnnotationProperty IRI="#labelCaption"/>
</Declaration>
...
<Declaration>
  <Class IRI="#LabelType"/>
</Declaration>
...
<Declaration>
  <Class IRI="#domain"/>
</Declaration>
...
<AnnotationAssertion>
  <AnnotationProperty IRI="#labelCaption"/>
  <IRI>#domain</IRI>
  <Literal datatypeIRI="&amp;rdf;PlainLiteral">Domain</Literal>
</AnnotationAssertion>
...
<SubClassOf>
  <Class IRI="#domain"/>
  <Class IRI="#LabelType"/>
</SubClassOf>
...
<Declaration>
  <ObjectProperty IRI="#hasprocessing-type"/>
</Declaration>
...
<ObjectPropertyDomain>
  <ObjectProperty IRI="#hasprocessing-type"/>
  <Class IRI="#Activity"/>
</ObjectPropertyDomain>
...
<ObjectPropertyRange>
  <ObjectProperty IRI="#hasprocessing-type"/>
  <ObjectMaxCardinality cardinality="3">
    <ObjectProperty IRI="#hasprocessing-type"/>
    <Class IRI="#processing-type"/>
  </ObjectMaxCardinality>
</ObjectPropertyRange>
...

```

## 5.6 Algorithm Complexity and Performance

The runtime of any type of reasoning (inference) algorithm is generally linked to the expressivity of the ontology language used. Hitzler et al. [77] note that “usually the higher expressivity comes at the expense of speed: the runtime of algorithms for automated inference tends to increase drastically when more expressive formalisms are used.”

Because the proposed semantic model makes use of cardinalities other than the basic cases of “0” and “1”, an OWL flavour of medium expressivity (OWL DL) must be used (in contrast to OWL Lite and OWL Full). The sublanguage OWL DL has a worst-case computational complexity NExpTime (solvable by a *non-deterministic* Turing machine in time  $O(2^{p(n)})$  for a polynomial  $p(n)$ ) [77]. Nevertheless, the cardinalities are not necessarily involved in any reasoning related to workflow composition and are only evaluated for user interface enhancement. Disregarding the cardinalities, OWL Lite can be used for complexity considerations. The worst-case computational complexity of OWL Lite is ExpTime (exponential runtime, solvable by a *deterministic* Turing machine in  $O(2^{p(n)})$  time).

The general theoretical complexity of semantic reasoning might only be of secondary importance for this work. More interesting is the performance of specific algorithms such as the label-based activity matching (Section 5.4.6). To match two objects with semantic labels, for instance, a maximum runtime of  $O(n * m * d)$  is required, where  $n$  and  $m$  are the label counts of the objects and  $d$  is the depth of the ontology (longest path from root to a leaf label type).

Table 6 contains the results of performance measurements for different actions. The execution environment was a 64 bit Windows PC with an Intel Xeon CPU at 3.3 GHz, 16 GB RAM, and Oracle Java JDK version 1.8 (with default settings).



Table 6 - Runtime measurements for the workflow system prototype

<b>Action</b>	<b>Average Time (ms)*</b>	<b>Standard Dev.</b>
Search for “manual” tools in the example activity repository (16 result items from 104 activities), including result rendering	35.4	0.9
Activity matching (source activity with two data ports, 4 labels) for 104 target activities, without rendering of results	4.2	0.1
Workflow concretisation (segmentation example, 3 abstract activities) selecting from 104 activities, without user interface operations	45.7	1.2
Text-based search (search term “image”) within all workflows (46 results from 126 workflows), including result rendering	68.8	1.2

(\*) The average time was calculated from five independent runs of the respective action.

## 5.7 Summary

A workflow system implementation based on the Java programming language was presented. The strategy was to include all features to compose and use workflows in an easy-to-use way (e.g. searchable repositories, semantics-based matching, and concretisation), but also to include all aspects supporting this, including workflow templates, workflow validation, and ontology management. This holistic approach, incorporating all mentioned aspects, can help to reduce the learning curve for the system. The users will only be exposed to relevant functions and user interface components (for instance when creating or editing an ontology internally as opposed to externally using Protégé, for example).

Building upon the design in Chapter IV, objective 4 from Section 1.4 is now satisfied. The presented prototype represents a complete framework for all relevant workflow-related tasks,

using new semantic approaches for automation and user support. This chapter also provides more details on the solutions for objectives 1. (modelling approach) and 3. (algorithms).

This is the end of the workflow system description. The next chapter focuses on the evaluation of the system.

## 6 Experiments, Use Cases, and Evaluation

The previous chapters described theoretical foundations, design, and implementation of an ontology for document image analysis and of a workflow system with semantic features. In this chapter, the validity, the possibilities, the advantages, but also the limitations of the proposed approach are explored, using real-world use cases from the domain of document image analysis (the system can be used for other domains – discussed in the last chapter, Section 8.5.1).

The main goal of this and the next chapter is to test the hypothesis by means of real-world data (objective 5 of Section 1.4). Using data from the target domain helps to establish more convincingly if semantic data and their use can aide users and make workflow systems easier to use.

All aspects of the proposed solutions were evaluated, the main ones being:

- The label-based ontology (see Chapter 3).
- Repositories (see Section 4.2).
- Workflow creation and management (see Section 4.3).

For a more relatable and praxis-oriented analysis, real data was favoured over synthetic data. To this end, the prototype workflow system was used to build up example data in the form of:

- A repository containing activities that represent software tools of the domain of document image analysis.
- Example workflows for common document image analysis tasks.

The data was then used to test the semantics-based features of the system but also complementary functionality.

The experimentation itself was thereby a valuable source of feedback for improving the models and software tools.

Before examining the workflow model and system, the ontology presented in Chapter 3 is evaluated in the following section.

## 6.1 Evaluation and Extension of the Ontology for Document Image Analysis

As discussed earlier in this Thesis, to enable a larger degree of automation in workflow creation, specific information must be incorporated into the workflow components. The solution proposed in this work uses semantic labels for activities and data objects. As has been described in previous chapters, the entirety of all types of labels for a certain domain represents an ontology. The usefulness of assistive and automated features strongly depends on the expressiveness of the label types. An ontology should therefore capture many different and, most crucially, relevant aspects of the domain it represents.

As mentioned in Section 3.4, a use-case centric evaluation approach (see [84]) is a valid and direct method to prove the applicability of an ontology. Accordingly, the aim of this evaluation effort (and of the wider research work) was to create a semantic model that has real-world relevance.

Workflow repositories are one main aspect of the proposed system (see Sections 2.4 and 4.2) as they allow semantic search and retrieval by both human users as well as by other features of the software framework (i.e. automation-related features). *Activity repositories* are specialised workflow repositories where each workflow contains, by convention, only one activity (a concrete atomic activity).

Building a semantically annotated activity repository serves two purposes:

- (1) Test the expressivity of the ontology: Can the concepts and terms of the ontology express all aspects of selected software tools (purpose, input data, and output data)?
- (2) Provide the basis for experiments regarding workflow creation: The activities are building blocks for concrete executable pipelines or scientific experiments.

Led by these arguments, it was decided to build up an activity repository referencing actual, existing software tools from the target domain. The process is described in the following subsection.

### 6.1.1 Building an Activity Repository

Over 100 software tools were identified for creating an example activity repository. Sources of information included:

- Reports from EU-funded projects (such as SUCCEED [94]).
- Dedicated websites (by tool creators).
- The tool framework of the Pattern Recognition and Image Analysis (PRImA) research lab at the University of Salford (an extensive collection of tools and data formats for document image analysis, see [100]).

The proposed model of a workflow system specifies an *activity* as having metadata, data ports, and semantic labels (for the activity itself and for each data port (see Section 4.1.1)). The task was therefore, based on the information above, for each software tool, to:

- (1) Create an atomic activity and add it to a repository (using the repository hub and the workflow editor).
- (2) Add metadata (description, version) and semantic labels (using activity-related taxonomies of the proposed ontology) to the activity object.
- (3) Create data input and output ports matching the capabilities of the tool.
- (4) Annotate the data ports with semantic labels (using the data-related taxonomies of the ontology).

As a reminder, the proposed ontology contains label types for activities and data objects covering the following categories:

- For activities: automation, licence, platform, domain, processing level, data transformation, adaptability, and maturity.

- For data: original source, age, production method, acquisition method, precision, content type, encoding, content of interest, granularity, condition, properties, and topic.

The semantic labels were extracted from the tool documentation / technical specification, considering the concepts from the ontology but also looking for aspects that could not be modelled (discussed in next subsection). Within the documentation of the Abbot tool (an XML conversion tool), for example, it is stated that:

*“This is pre-release software. It may fail to compile. It might have undocumented features. It certainly has unimplemented features. It definitely has bugs.”*

In such case, the activity can be labelled as “Maturity - Experimental”.

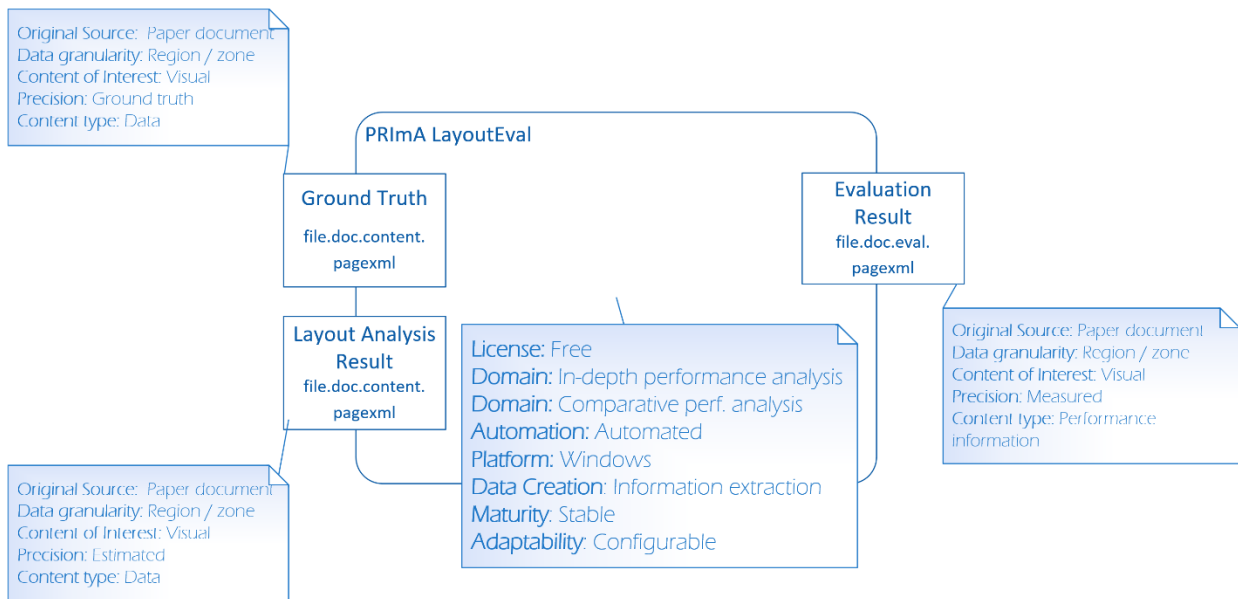


Figure 78 - Example of an atomic activity in UML notation (including labels for data ports and activity)

Figure 78 shows an example activity in detail (using the UML notation introduced in Section 4.1.1 extended by semantic labels in blue annotation boxes), including data ports and labels. The respective software tool is the PRImA Layout Evaluation Tool for evaluating the performance of page segmentation methods. For clarity, labels are written in a shortened

form and not their full representation (which is the path within the ontology starting from the root label type and ending at a leaf of the corresponding taxonomy).

Notably, one label root type (“Domain”) is used several times for one object. This shows the usefulness of cardinalities greater than one within the semantic model. This was introduced as *label slots* in Section 4.1.5. Multiple labels from one label type taxonomy can thereby be used to annotate an object (the labels have different types are part of the same sub-tree in the ontology).

The fact that both input ports have the same data type (“file.doc.content.pagexml”) underlines the importance of semantic annotation. Without additional information, an automated method would not be able to distinguish between the two inputs. This information is represented through the label types “Precision” (“Ground truth” vs. “Estimated”).

An important fact is that software tools and activities are not strictly one-to-one relations but rather one-to-many relations. This means, if a tool has multiple distinct functions, it can appear in multiple activities. These tools are applicable to different use scenarios and accordingly support several output data types. Examples are ABBYY FineReader Engine (used as OCR engine or for image pre-processing) and Aletheia (used for creation of different types of ground truth). In these cases, the specific application is reflected by the activity captions.

Table 7 – Example activities (by category and in alphabetic order) provides an overview of the collected software tools. A more detailed list can be found in Appendix C – Software tools used for activity repository.

Table 7 – Example activities (by category and in alphabetic order)

Conversion	Annotation	Acquisition	Evaluation	Information Management	Text Processing	Image Processing		Analysis / Recognition	
abbot	Aletheia	Document capturing by camera	HOCR Eval	Agora	Apache OpenNLP Named Entity Recognition	ABBYY Image Pre-processing	PRImA Region Extraction	ABBYY Business Card Reader	OCROPUS OCR
PRImA Table Exporter	From The Page	Document scanning	ISRI Evaluation Tool	File Analyzer	Apache OpenNLP Sentence Detector	ABBYY Binarisation	PRImA Text Line Extraction	ABBYY Block Segmentation	OneNote Handwriting Recognition
PRImA Text Extraction	GEDI	LayoutEval Profile Creation	LayoutEval	jMet2Ont	ASV Toolbox - Jlanl	Aletheia Dewarping Ground Truth Creation	PRImA Word Extraction	ALTO-Edit	PRImA Table Classifier
PRImA PAGE to SVG	Islandora		MILE OCR Performance Evaluator	MapForce	Berkeley Parser	Cam-Scanner	Sauvola Binarisation	AWE Layout Editor	PRImA Text Line Segmenter
PRImA Page Conversion	PRImA Crowd Prototype	<b>Visualisation</b>	NCSR OCR Evaluation Tool	Metadata Extraction Tool	Brevity	Document Deskewer	Scan Tailor	Clara OCR	PRImA Typewritten OCR
PRImA PAGE to PDF	PRImA JPageViewer	Layout Analysis Performance Visualisation	PRImA Dewarping Evaluation	Pandoc	CLAWS	Hectography Foreground Extractor	tifftool	ABBYY FineReader Engine 11	PRImA Word Segmenter
	PRImA Page Feature Extractor	PDF-XChange Viewer	PRImA Page Validation	PRImA JFeature-Extractor	cue.lang-uage	Hot Metal Font Enhancer	Unpaper	Fraunhofer News-paper Segmenter	Shape-Catcher
	Transcript	PRImA PAGE Visualisation	PRImA OCR Evaluation Tool	PRImA Table Cell Post-Processor	Graph-based Dependency Parser	GIMP	Wavelet image de-noising	Functional Extension Parser	SharpEye Musical Score Recognition
	Word-Freak		Text and Error Profiler	PRImA Page Metadata Extractor	IMPACT Spelling Variation Tool	Image Magick		Goggles	Tesseract 3.03
					MALLET Document Classification	NCSR Binarisation		Ground Truth Maker	Type-Wright
					Monty-Chunker	NCSR Border Removal		GTText	ZBar
					Monty-Lemmasiser	NCSR Page Curl Correction		PRImA Layout Aligner	
					Monty-Tokenizer	Otsu Binarisation		Lios	
					PRImA Text Normalisation	PRImA Dewarping		NCSR Character Segmentation	
					Stanford Parser	PRImA Glyph Extraction		ocrad	



## 6.1.2 Discussion

The semantic annotation of the 104 activities (representing software methods) was a viable, independent way to evaluate the ontology for completeness because the tools are used to perform common tasks in document image analysis, but they were not used as information source during the first ontology life cycle. The specifications of the software tools contain information on input and output *data*. That information could be encoded well using the existing label types. The available tool descriptions, reflecting *activity*-related concepts, could also be modelled well using the available label types. All software methods could be semantically annotated to a high degree with respect to data ports, functionality, and metadata. About 1,700 labels were used, averaging to ca. 16 labels per tool/activity.

Occurrences where certain information could not be reflected using the proposed semantic model are discussed in the remainder of this subsection.

### 6.1.2.1 *Speed and Performance*

Several tool descriptions state performance-related characteristics. Abbot, for example, mentions:

*“Abbot is designed to be furiously fast (it will automatically parallelize the conversion across  $n$  processor cores)”.*

Aspects like this are not part of the ontology and they are difficult to quantify. General information could be included in future extensions of the ontology (“Parallelisation” or “Multi-threading” for example). Chapter 8 contains more thoughts on this subject.

### 6.1.2.2 *System Requirements*

Some software tools (mostly the commercial ones) list detailed system requirements such as memory and hard disk space. At the moment the ontology only offers platform-related labels. Further labels could be added in the future, but this must be considered carefully since the existence of certain label types may raise expectations by users. If only a fraction of all

activities is labelled with system requirements, the other tools might not be discovered during a search because this information was not available at repository entry time.

#### 6.1.2.3 Method / Processing Details

The ontology is a model of the real world and, as such, a simplification (by definition). The level of detail must be chosen as a trade-off between model complexity / usability and coverage of the target domain. The ABBYY Image Pre-processing tool, for example, encompasses deskewing, text line straightening, photo correction, cropping, colour correction and more. Currently, the ontology only allows for a depth such as “Activity domain – Computing – Visual computing – Image and video processing – Pixel-based”. Sub-types can be added without breaking the compatibility of existing workflows, but, as mentioned before, this must be considered carefully. A certain level of abstraction can help to make the system easier to use. The exact activity / software features can be incorporated in the workflow description field.

#### 6.1.2.4 Algorithm Design

One major resource for creating the presented ontology were IJDAR journal papers (International Journal of Document Analysis and Recognition). Therefore, a significant amount of algorithm-related terms was collected during the conceptualisation stage. Most of these terms were intentionally left out from the final ontology because they have little significance for the target application – the composition and management of workflows. One example originates from the description of the AGORA tool:

*“The algorithms involved in AGORA use two maps to segment noisy images: a **shape map** that focuses on connected components and a **background map** that provides information on white areas corresponding to block separations in the page”.*

However, certain features of algorithms and methods are part of the ontology, if they have an impact on the applicability/use of activities, for example: supervised or unsupervised learning.

#### 6.1.2.5 *Documentation, Activity and Community*

One real-world aspect that is not captured well by the ontology is the degree of available user documentation, project activity, user community-related points (e.g. can help be expected when needed) and similar characteristics. These can be a factor when choosing between several tools that have the same range of functionality. The Aletheia system, for example, provides an extensive user guide, several case studies and video tutorials. In addition, it is under active development, increasing the chances of problems being fixed at some point, if they arise. Existing labels falling in this category are grouped under *Software Licence* and *Maturity*.

#### 6.1.2.6 *Multifunctional Software Tools*

It was mentioned before that some software tools appear in different activities of the repository because they offer significantly different functions. Other tools have a similar brevity of features. CamScanner, for example, includes mobile document scanning, image correction and enhancement, and text recognition. An alternative to adding two or more activities is to add *one* activity with multiple output ports, each labelled semantically for one main function of the software. When integrated in a larger workflow, the user or the system can select the output port which fits best for the task at hand. The CamScanner activity was consequently created with two output ports: “Image Scan” (including the image enhancement results) and “OCR Result”. It should be noted that this solution is only viable if the software tool can perform these actions in one go (otherwise multiple activities are required, or a more complex workflow model would have to be developed).

The next subsection lists cases where the review lead to improvements of the ontology.

### 6.1.3 Extension of Ontology

The purpose of the evaluation was to validate the ontology but also to improve it where necessary or useful. The following modifications were made:

- Insertion of “Natural language” as subtype of “Data properties – Text-related – Language”: This was done to be able to label certain input data as containing natural language in general (e.g. for natural language processing methods). Other languages could be formal languages such as programming languages.
- Addition of label type “Tabular” as subtype of “Content Encoding - Structured”. This can be a useful cue to find a suitable converter tool.
- Addition of label type “Layout analysis” as subtype of “Activity Domain – Computing – Visual computing – Content analysis and recognition”. This was necessary for methods that perform page layout analysis and text recognition but only output the recognised text (e.g. “ocrad”). The information that the layout of the input image is taken into consideration is important as it can have a significant impact on the OCR performance (due to column separation etc.).
- Addition of label type “Barcode / QR code” and subtype of “Content of interest – Visual – Graphical”. There was no explicit way of denoting barcode-encoded data input (as applicable for the “ZBar” tool, for instance).

Apart from the aforementioned limitations, the properties of the 104 activities (involved in this research) could be reflected using the semantic labels from the proposed ontology for document image analysis. In total, 1,696 labels were assigned within the activity repository (at the time of writing).

The next sections focus on workflow search and composition, applying the presented activity repository.

## 6.2 Evaluation of Search Functionality in Repositories

Search for items in a repository is a main feature of the Repository Hub of the proposed system. Users can search for workflows, data tables, or activities (needed during workflow creation). A successful search presents the user with items that are relevant in the respective situation.

The search functionality of the workflow management system was analysed using the newly created activity repository as a data basis. As described in Section 5.3, the search interface provides filters for all label types that are used within the workflows of the repository (the activity repository contains workflows with one activity each), here representing real-world software tools (see Figure 79 and Figure 43 on page 97). The filters (represented by checkbox controls) are thereby grouped into:

- Workflow features (activity labels).
- Input data (input port labels).
- Output data (output port labels).

In addition, a grouping according to taxonomies (root label types) is applied (for example “Automation” and “Licence” as seen in Figure 79). The hierarchical structure of the ontology is represented through indentation of the filter controls (checkboxes).

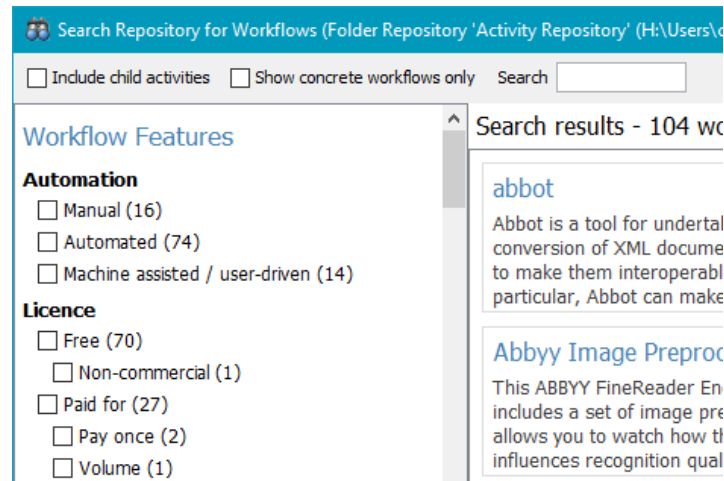


Figure 79 - Search filters in Repository Hub (root label types in bold; child label types indented according to depth in hierarchy; numbers reflect the number of labels found within the current repository)

The goal was to find an objective measure for the efficiency of the interface in combination with the ontology (for document image analysis). The *number of user interactions* to achieve a useful search result can be seen as such a measure. In context of the search dialogue, this can be interpreted as the number of required filter selections to

sufficiently reduce the search result (from a large number of results to a small number of results).

A specific value for the number of search result items that are considered sufficient for a user had to be chosen. This was done in the following manner.

Let  $U$  be a hypothetical user of the search interface  $I$ . Let  $n$  be the number of search results,  $n_{max}$  the maximum number of search results, and  $n_{suff}(U)$  the number of search results that satisfy user  $U$ .

The user interface  $I$  presents  $U$  with  $n_{vis}$  number of items at a time. Finally, the number of interactions with  $I$  by user  $U$  is called  $a_{UI}$ . See Figure 80 as illustration.

It was decided to set  $n_{suff}(U)$  equal to  $n_{vis}$  which is 14 by default. The reduction of search results is thus defined as “sufficient” if the result items fit the default window size.

The evaluation then determines the maximum number of interactions  $a_{UI,max}$  that reduce  $n$  from  $n_{max}$  to  $n_{suff}(U)$ . In practical terms this means: How many checkboxes does the user have to tick, to be able to see the filtered search result at one glance (without need to scroll)?

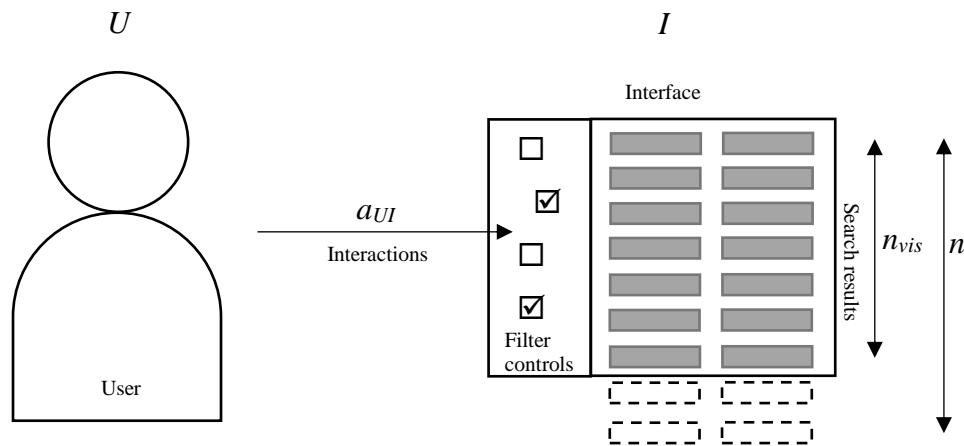


Figure 80 - User interaction with search interface

Note that this is an evaluation with strict constraints for the purpose of finding a measure and achieve reproducible results. Actual users can resize the search dialogue to fit more results or scroll to view more items.

To find the maximum number of required checkbox selections  $a_{UI,max}$  (equal to mouse clicks), at each step the checkbox with the *highest* number of corresponding labels was selected (i.e. the worst case). Label types that represent an abstract grouping and have no siblings (“Computing” in the “Activity Domain” taxonomy, for instance) were left out because their only function is to future-prove the ontology. Only child label types of these abstract parent types were used to annotate activities.

The experiment was conducted for all label types at once but also for each major label group individually (workflow features, input data, output data). Table 8 shows the outcome. The list of workflows (representing software tools in this case) can be narrowed down with only a few steps. In the worst case, the user has to tick nine checkboxes.

The average number of selections will be considerably lower in real-world use. In some cases, just one click is sufficient (the label “Performance Evaluation” represents nine workflows, for instance).

*Table 8 - Results of workflow search experiment*

<b>Label Group</b>	<b>Maximum Number of Checkbox Selections</b>
	<i><math>a_{UI,max}</math></i>
Workflow features	5
Input data	5
Output data	5
All	9

In general, the search by semantic label is a valid approach for finding activities or workflows. The concept of checkbox-style filters with attached counts presents a hierarchical straightforward method to narrow down search results (not dissimilar to systems used in online datasets or online shops for example).

A future implementation of the workflow system will require a complete user interface design process and possibly an evaluation involving users from different target groups.

The following subsection contains an analysis that was performed to gain some insight into the distribution of labels within the activity repository.

### 6.2.1 Activity Repository Decision Tree

To analyse the labels that have been assigned to the activities in the example repository, a method known from data mining was used: decision tree creation.

The labels of all activities can be interpreted as binary attributes (features) of the form:

$$attr_i = \begin{cases} \text{TRUE, if activity has label of type } i \\ \text{FALSE, otherwise} \end{cases}, \forall i \in L$$

Where L is the set of all labels that have been assigned to one or more activities in the repository. These attributes can be arranged in a table with one column per label type and one row per activity (104 in the example repository). The label types are thereby handled separately for activity-related, input-data-related, and output-data-related labels, leading to 152 attribute columns for the example repository.

Decision trees represent a divide and conquer approach (see [101]). The objects of interest are divided step-by-step into smaller sets. How many decisions have to be made depends on the quality of the respective attributes (the decision tree nodes). Decision tree algorithms try to create the optimal tree by ordering the attributes in a specific way. The ID3 algorithm uses a measure called Information Gain (IG) to rank the attributes according to how well they help to divide the set of activities (for more information see [101]).

The open source data mining tool WEKA [102] includes an implementation of ID3. Figure 81 shows the decision tree creation step and Figure 82 shows the first four levels of the resulting tree. WEKA also offers an attribute selection algorithm based on the information gain. Table 9 shows the 20 most dividing label types and the 20 least dividing, according to the IG measure.

From the results it can be observed that certain label types lead to a quick reduction of search results. If the target platform of a workflow is known to be a Windows operating system, starting a search by selecting “Activity – Platform - Windows” can make the subsequent search easier.



In the following section, features related to workflow composition are evaluated, using an example workflow for a common task in document image analysis.

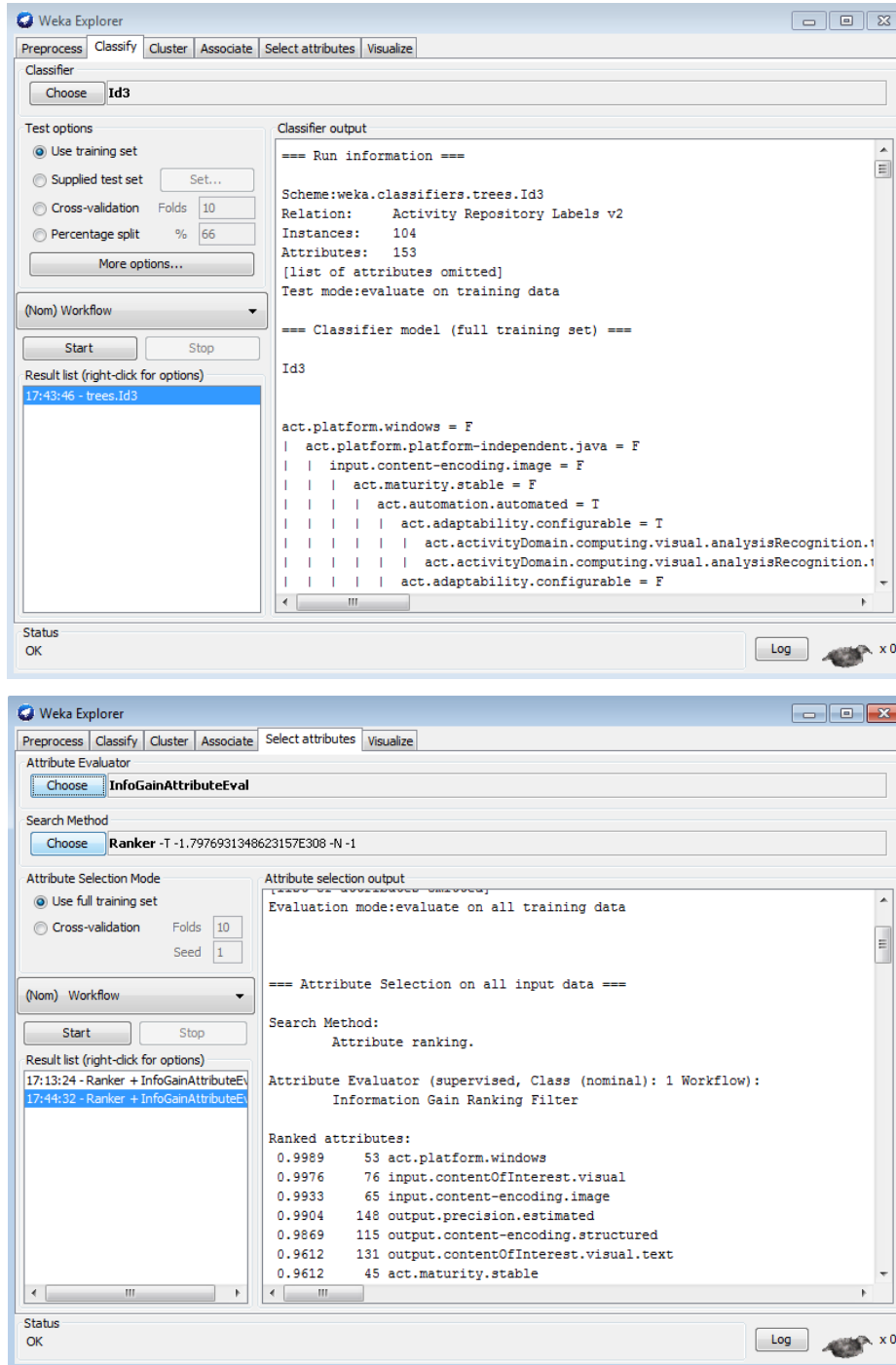


Figure 81 - Decision tree creation (left) and information gain ranking (right) in WEKA software tool

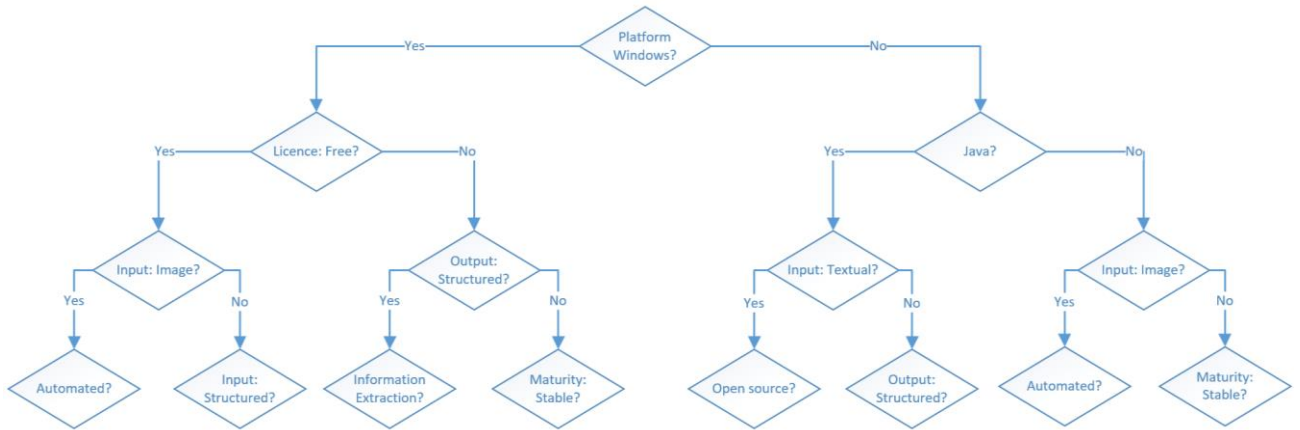


Figure 82 - Decision tree for activity labels (partial)

Table 9 - Label types sorted by information gain (20 best and 20 worst ranked label types)

Score	Label type
0.9989	activity.platform.windows
0.9976	input.contentOfInterest.visual
0.9933	input.content-encoding.image
0.9904	output.precision.estimated
0.9869	output.content-encoding.structured
0.9612	output.contentOfInterest.visual.text
0.9612	activity.maturity.stable
0.9391	input.contentOfInterest.visual.text
0.9215	activity.license.free
0.9215	input.content-encoding.structured
0.8667	activity.automation.automated
0.8539	output.content-type.data
0.8404	activity.adaptability.configurable
0.8113	output.contentOfInterest.visual
0.8113	activity.processingLevel.high-level.classification
0.7957	activity.dataTransformation.enhancement
0.7793	activity.maturity.experimental
0.7793	activity.dataTransformation.extraction
0.7623	activity.license.paid-for
0.7623	activity.platform.platform-independent.java

Score	Label type
...	
0.0782	output.granularity.logical.document-related.article
0.0782	output.data-attributes.text-related.language
0.0782	input.topic.economy.financial
0.0782	input.production-method.manual
0.0782	input.production-method.machine.typewritten
0.0782	input.production-method.machine.printed.typeset
0.0782	input.content-type.metadata
0.0782	input.condition.production-faults
0.0782	activity.processingLevel.high-level
0.0782	activity.platform.mobile.ios
0.0782	activity.license.paid-for.volume
0.0782	input.content-type.model
0.0782	output.production-method.machine.printed.typeset
0.0782	input.contentOfInterest.visual.graphical.barcode
0.0782	input.originalSource.captured.scenes
0.0782	input.originalSource.produced.physical.paper.newspaper
0.0782	input.granularity.logical.document-related.document
0.0782	input.data-attributes.text-related.visual.complex-background
0.0782	input.data-attributes.text-related.visual.multi-font
0.0782	input.contentOfInterest.visual.composite.music

## 6.3 Evaluation of Workflow Composition Functionality

This section explores the assistive and automated features of the prototype workflow composition software using a typical use case from the domain of interest. While this section presents a workflow of low complexity (to focus more clearly on the evaluated tasks), Chapter 7 presents a more complex case study.

Image segmentation is a fundamental task in layout analysis “that, given the source document representation (vector or raster), returns a partition of its area into portions of content representing significant pieces of the layout.” [7] Segmentation is often required because other tasks require (or work better with) specific page layout objects such as blocks or text lines.

First, a workflow for segmentation is introduced, followed by the application of assistive and automated features of the prototype system.

### 6.3.1 Workflow for Page Segmentation

A workflow can either be explicit (concrete) and directly executable or it can be abstract, where certain activities within the workflow are generic placeholders for specific types of software tools. The latter facilitates reusability of workflows since only the general structure (control flow) is fixed and details can be filled in using different software methods (creating

different specialised workflows). Section 5.4.5 introduced this concept as *workflow templates*.

A template for page segmentation (a common task within document image analysis) was created, containing three core activities: *segmentation* itself but also *image conversion* and *pre-processing*. Figure 83 shows the UML notation of the example workflow.

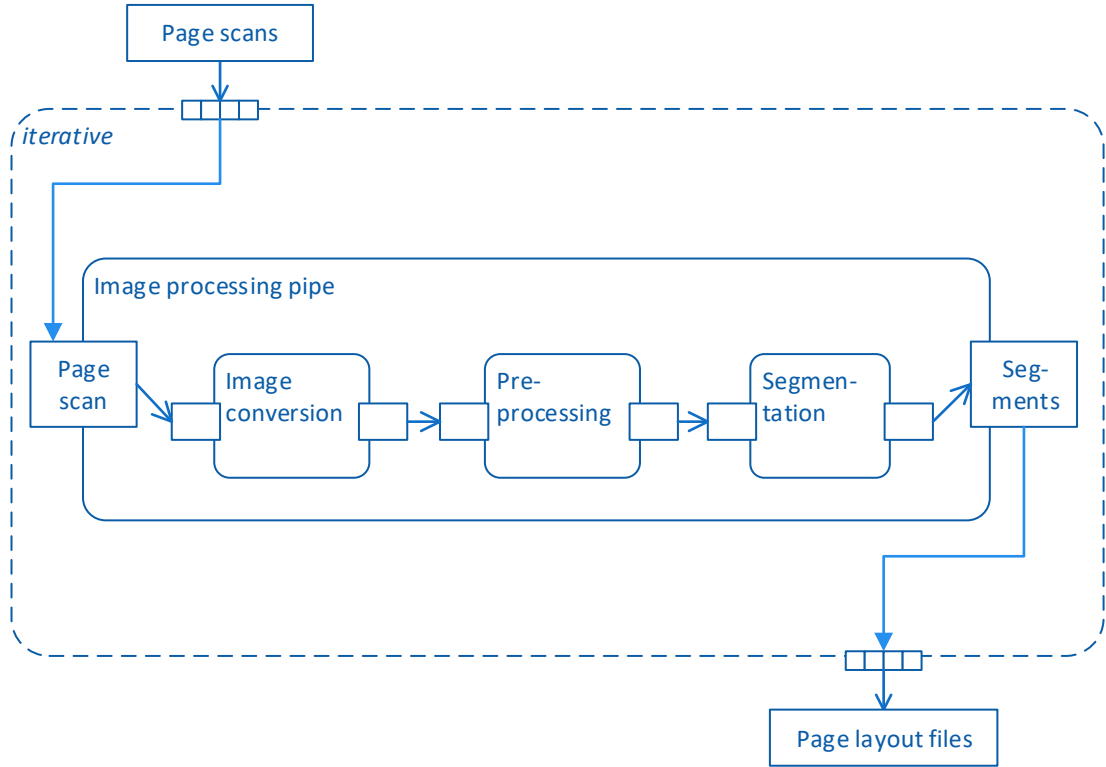


Figure 83 - Example workflow template with a for-loop and directed graph activity (UML notation)

In the workflow, document images (page scans) are segmented one by one, producing result files containing the page layout (segments). Three abstract atomic activities are used as placeholders for image conversion, image pre-processing, and segmentation tools (in that sequence). Metadata and semantic labels were specified accordingly (see Table 10 for used labels). The complete template therefore consists of five activities: a for-loop activity, a directed graph activity, and the three placeholders.

The following subsections describe how to create a concrete workflow (and instance of the template) from the template within the workflow editor of the system prototype.

Table 10 – Labels used for the example workflow (limited to objects relevant to the case study)

<b>Activity</b>	<b>Labelled Object</b>	<b>Semantic Labels</b>		
		<b>Label Root</b>	<b>Label</b>	
<b>Loop</b>	Activity	Data creation / transformation	Segmentation / tokenisation	
		Activity domain	Layout analysis	
	Input port	Data granularity	Page	
		Content encoding	Raster image	
		Content type	Data	
		Content of interest	Visual content	
<b>Image Conversion</b>	Activity	Data creation / transformation	Conversion	
		Automation	Automated	
	Input port	Content encoding	Raster image	
		Output port	Content encoding	Raster image
	<b>Pre-processing</b>	Activity	Data creation / transformation	Enhancement
			Automation	Automated
		Input port	Content encoding	Raster image
Output port			Content encoding	Raster image
<b>Segmentation</b>	Activity	Data creation / transformation	Segmentation / tokenisation	
		Automation	Automated	
		Processing level	Classification / recognition	
	Input port	Activity domain	Layout analysis	
		Data granularity	Page	
		Content encoding	Raster image	
		Content type	Data	
		Content of interest	Visual content	
		Output port	Data granularity	Region / zone
			Content encoding	Structured
			Content type	Data
			Content of interest	Visual content

### 6.3.2 Replacing an Activity

Assisted activity replacing is an important feature of the proposed system. It uses the semantic label-based matching algorithm introduced in Section 5.4.6.

To demonstrate the effectiveness of the matching algorithm in context of the example workflow and the activity repository from Section 6.1.1, two tests were carried out for the *Replace Activity* function. First, *including* the semantic labels for the matching and, second, *excluding* the semantic features.

The Replace Activity function was triggered for the atomic activity called “Pre-processing” (the other two atomic activities can be handled similarly). The corresponding dialogue offers (Figure 84):

- Source repository selection (the activity repository from section 6.1 was chosen here).
- Label filter controls (not required in this experiment).
- Matching controls (parameters for the matching algorithm).
- Result item panel (workflows with associated match scores).

With the “Match labels” option enabled, the matching algorithm returned nine items with a score of 100%, three items with a 92% score, and the remaining activities with scores ranging from 83% down to 0%. All nine activities with the maximum score represent relevant image pre-processing software tools that would be applicable to the workflow at hand. The three “runner-ups” are pre-processing activities but are based on an interactive user interface and do not match the “automated” label of the template.

Figure 84 shows the matching result and the details for one match with 92% score. The user can then browse the results and make a decision (to either use one of the returned activities, refine search results, or remove the placeholder activity from the workflow).

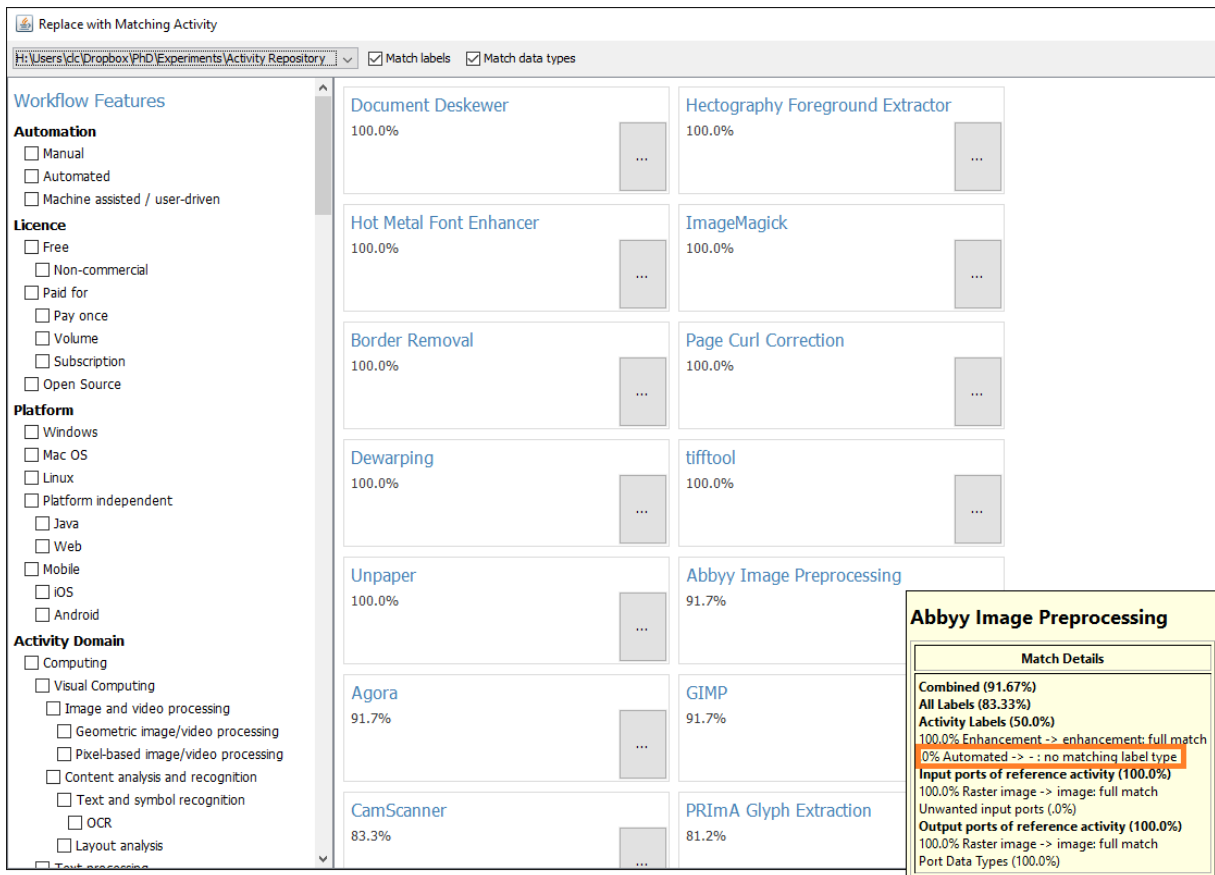


Figure 84 - Activity matching result with details (missing label for "automated" highlighted)

The test was repeated with the “Match labels” option disabled. The matching algorithm then only uses *data types* of input and output ports to calculate the match score. Matching only by data type resulted in 27 items with a score of 100%. Considering the previous finding (9 items with 100%), two thirds of these activities are (semantically) unsuited for the workflow template. This highlights the usefulness of semantic information (match results are more relevant).

The step of replacing the placeholder activity with a result item is straightforward (performed via mouse click on result item). Aligning data ports is not necessary for this example since both activities have exactly one input and one output port. The outcome, shown in Figure 85, is as expected (the abstract pre-processing activity is replaced by the Border Removal activity in this example).

The experiment showed that the semantic annotation and the use of this information for the task of replacing an activity represent an effective and useful enrichment of the workflow system. Users benefit by being presented with search results in an order more relevant to the context. Semantics-enabled match scores better reflect the applicability of activities than match scores that are based on data type information only.

It should be noted that using the semantic labels during matching does not reduce the number of search results. It only affects the ordering and visualisation (match score captions). The user can still navigate to all result items.

The next subsection shows how to let the workflow system choose activities to create a viable workflow from the given template.

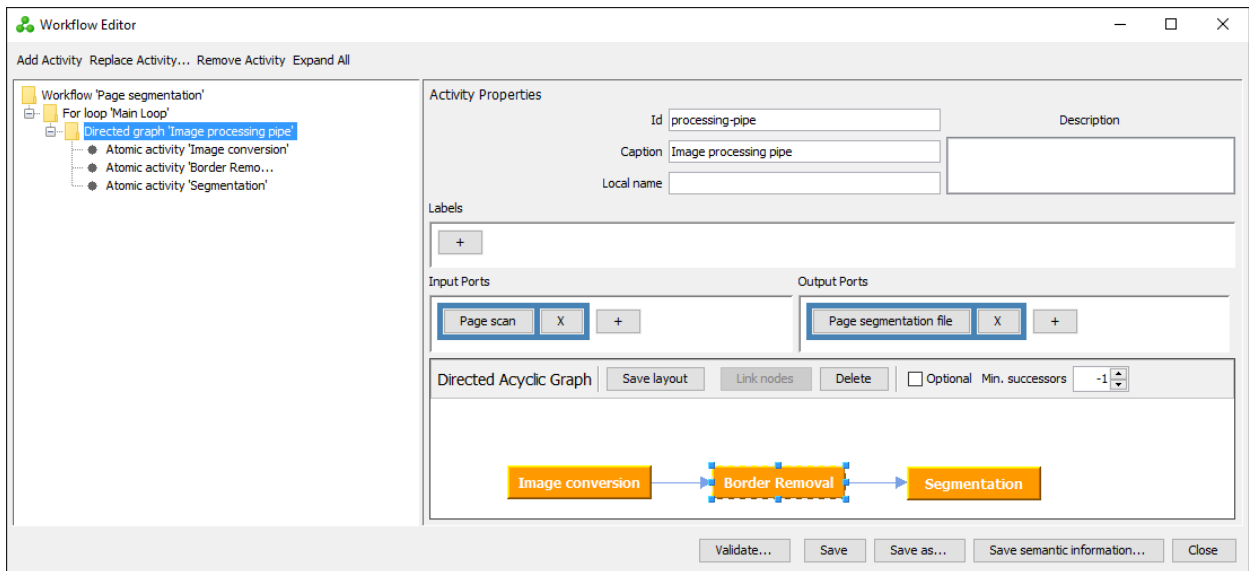


Figure 85 – Pre-processing activity of workflow template replaced with "Border Removal" activity

### 6.3.3 Automated and Assisted Workflow Concretisation

The previous subsection described a semi-automated feature of the workflow system. Now, the prototype's potential of a *fully-automated* method is explored: instantiating a workflow template (i.e. concretisation, see Section 4.3.3).



A workflow is called *abstract*, if at least one of its child activities is abstract (see Section 4.1.4). The same principle applies to control flow activities such as loop and graph activities. Therefore, the abstractness can be traced to the lowest-level activities (leaf nodes in the activity tree). The task of concretisation can be summarised as replacing all abstract leaf activities (placeholders within a template) of the abstract workflow.

The workflow editor of the prototype system shows a “Concretise” button if the current workflow is abstract (see also Section 5.4.8). The corresponding dialogue, shown in Figure 86, contains options to:

- Select a repository that is used as activity source (containing concrete atomic activities).
- Choose between fully automated concretisation and an interactive (assisted) mode.

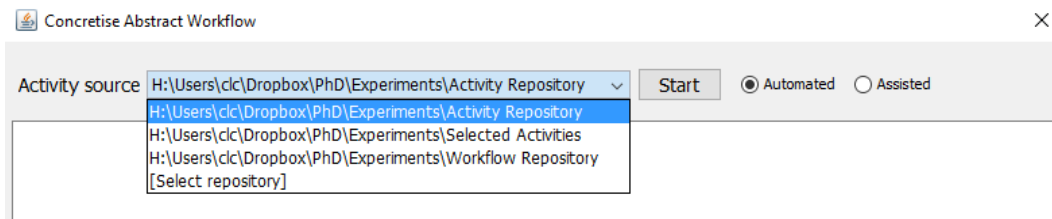
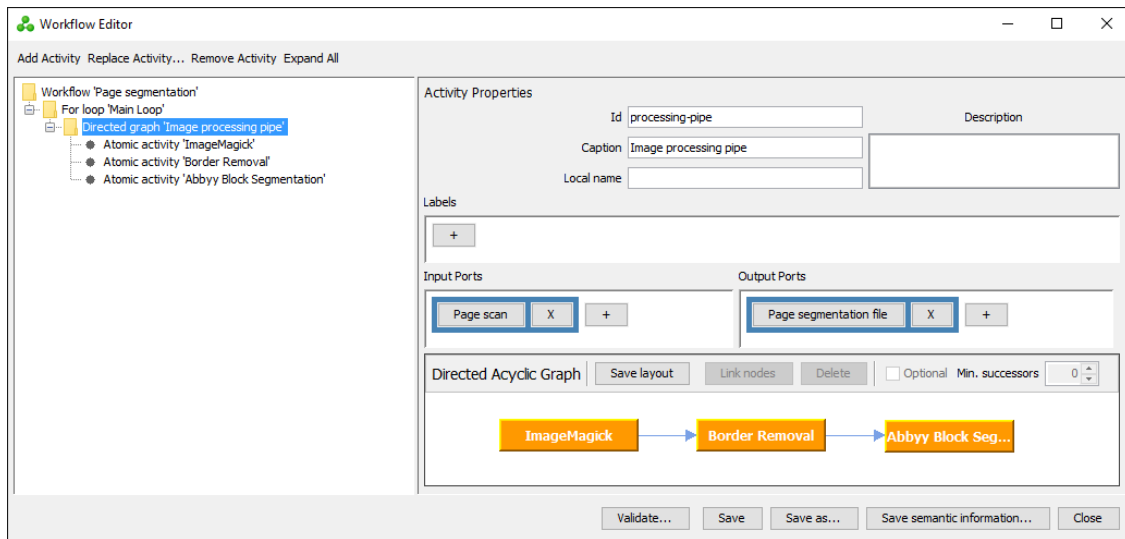


Figure 86 - Setup of workflow concretisation

Performing the automated concretisation using the example activity repository from Section 6.1 results in a valid workflow (Figure 87). Nevertheless, due to the size of the repository, there are many valid combinations of activities. In a future iteration, more data could be gathered to help to decide for an activity. If a workflow repository with a substantial number of templates is available, for instance, usage data (count, common combinations with other activities) can form the basis for a more informed decision. A similar solution would involve the extension of the ontology to be able to add more fine-grained semantic information to the workflow templates and candidate activities in the repository. The advantage of this solution is that it does not require the extension of the prototype system itself.



```
'Image conversion' replaced with 'ImageMagick'
'Pre-processing' replaced with 'Border Removal'
'Segmentation' replaced with 'Abby Block Segmentation'

Successful. The workflow is now executable.
```

Figure 87 - Result of automated workflow concretisation (top: three concrete atomic activities; bottom: concretisation log)

Another alternative to resolving ambiguities is with user interaction. The “Assisted” setting allows the user to resolve conflicts on-the-fly. The algorithm in Listing 3 (page 118) contains a call-back function that is called if multiple matching concrete activities are found that could replace the current abstract activity. In that case, a dialogue to refine the activity collection is shown (Figure 88), containing the following elements:

- Top: A message stating which abstract activity presents a problem and a description of the problem. The user can also check which activity placeholders have already been replaced by which new activities.
- Left: Filter controls to narrow down the list of activities. Only filter check boxes for the labels used in the activities at hand are shown, limiting the amount of filter options significantly (making it easier to spot relevant differences).
- Right: Grid view of all matching activities with their description and data port details (tooltip).

The user has a choice of selecting one activity (by clicking on it) or to use the filter controls to reduce the number of matches and let the tool make the final decision (“Continue” button).

The user interaction is also invoked if matching activities are found but the match score is lower than the strictness threshold. In such a case, the user is asked for confirmation.

The next subsection describes how to achieve a baseline quality of the created workflow using the validation feature of the system.

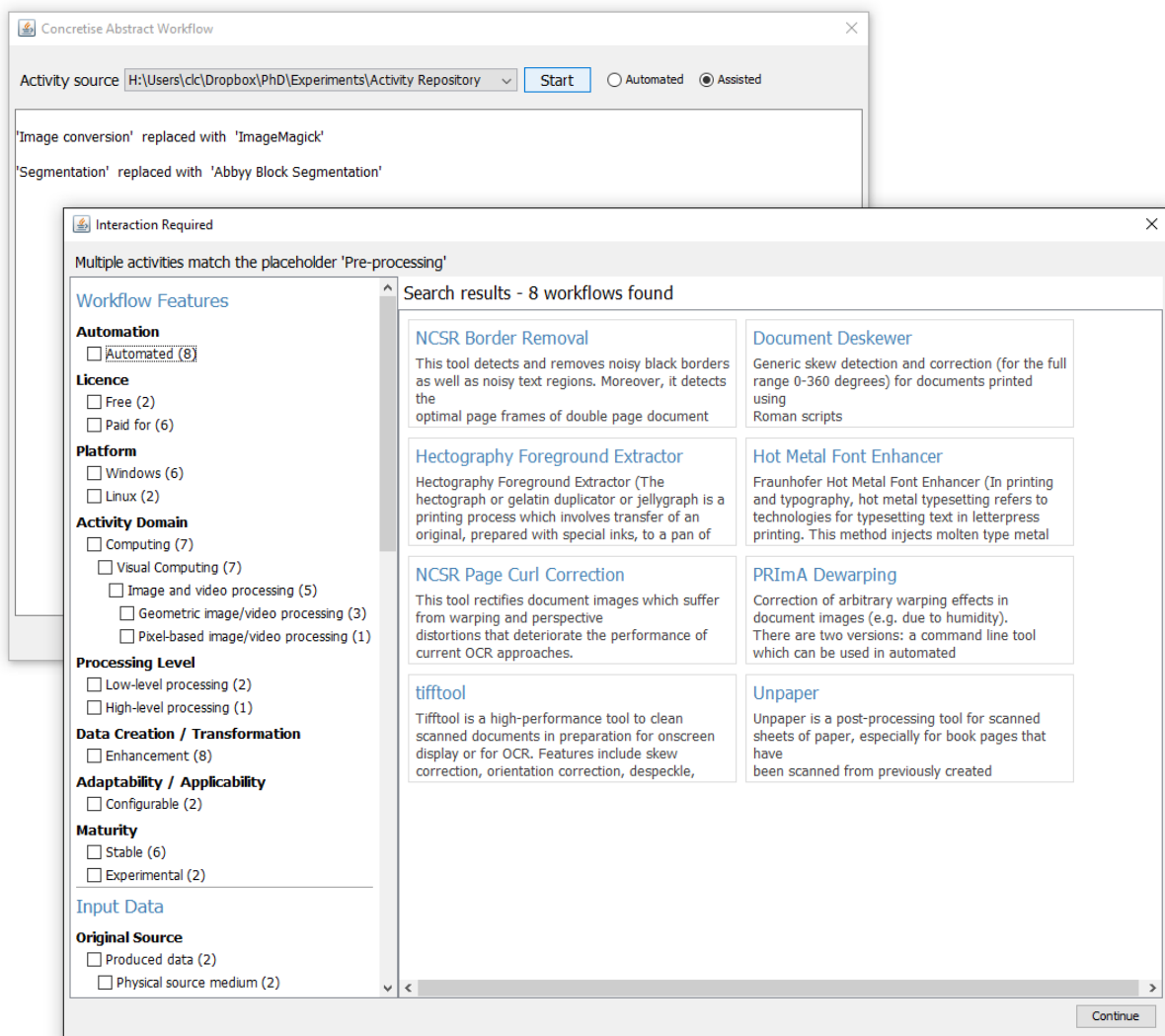


Figure 88 - Interactive workflow concretisation

### 6.3.4 Evaluation of Workflow Validation

Workflow validation (see subsections 4.3.2 and 5.4.7) is intended to aid the user to produce a workflow of good quality. To evaluate the validation functionality of the system prototype, two tests were performed. First, the example workflow template for page segmentation (see subsection 6.3.1) was validated and the results examined. Second, cycle detection was tested using an additional example workflow.

#### **Validation of Segmentation Workflow Template**

When applied to the example template (the user triggers the validation in the Workflow Editor), several problems are reported. Figure 89 shows selected messages and the corresponding panel with options to resolve the issues. If applicable, the workflow editor (open at the same time) highlights the workflow component that causes/contains the problem.

Resolving basic issues can help to prevent more complex problems at a later stage (which might be difficult to trace back to the source). In addition, completing all captions and descriptions will help other users who will use the template in the future.

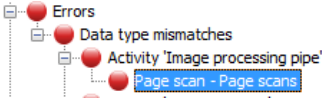
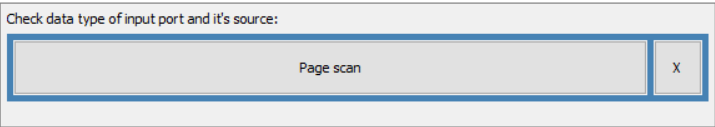
The issues reported for this specific workflow template (page segmentation) are listed and discussed below:

- *Mismatch* between the *data types* of two connected ports: The example shown in Figure 89a is caused by one of the ports having no data type specified and the other port using “file.image”. It is not always desirable to narrow down the data type of a template (more general application). However, limiting the type to any type of *image* seems reasonable and both data ports should be aligned in this sense.
- *Semantic mismatch* between connected ports: Figure 89b shows the report for two connected ports that do not match well on a semantic level. The validation message provides some details on which labels are involved. The user can solve this problem in the workflow editor. Labels for similar data objects should be assigned in a consistent way. Future implementations of the editor could include a feature to aid the user in this respect (e.g. copy labels from one port to a connected port).
- *Abstract workflow*: This message (see Figure 89c) is not relevant for the current example since a workflow template is abstract on purpose. Nevertheless, the user

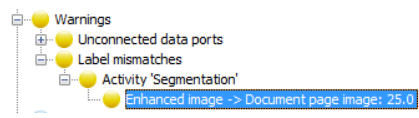
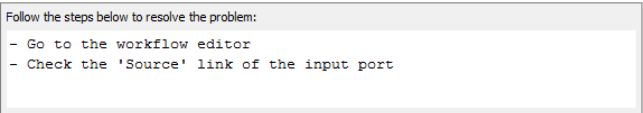
might be working on creating an executable workflow and simply forgot to concretise one abstract child activity.

- *Missing activity (or workflow) data*: Figure 89d show the message and correction panel which is shown if data fields such as caption or description are blank. As in software development and other domains, proper documentation is a crucial factor concerning the quality of the software (a workflow in this case). The missing data can be entered by the user directly in the correction panel.
- *Missing data type*: The message shown in Figure 89e is only shown for non-abstract (i.e. concrete) activities. To demonstrate the validation for this case, the workflow was concretised using the automated method described in Section 5.4.8. The correction panel contains controls to specify the data type(s) for the respective port using the dialogue that is also available in the workflow editor.

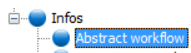
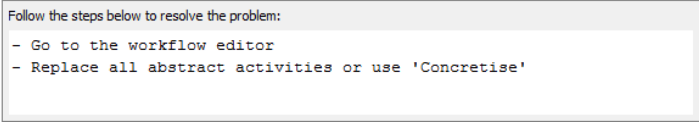
Data type of input port source doesn't match input port data type.

a)  

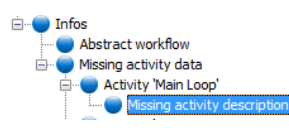
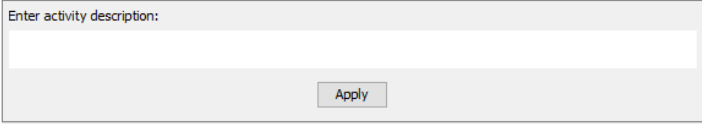
.0% Page <-> - : no matching label type  
 100.0% Raster image <-> image: full match  
 .0% Data <-> - : no matching label type  
 .0% Visual content <-> - : no matching label type

b)  

The workflow has at least one abstract activity and cannot be executed.

c)  

No description has been specified for the activity.

d)  

No data type defined

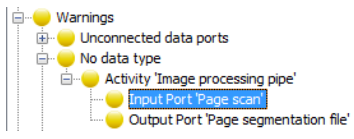
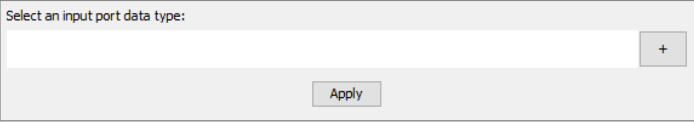
e)  

Figure 89 - Selected validation messages for the segmentation example workflow (a: data type mismatch; b: label mismatch; c: abstractness; d: missing description; e: missing data type)

## Test of Cycle Detection

The validation method can also find cycles in directed graph activities. To test the cycle detection, the graph activity of example workflow was modified to intentionally connect the segmentation output back to the image conversion activity. Figure 90 shows the result of the workflow validation (the cycle was detected and reported). Currently, this can only be resolved by the user (by deleting edges in the directed graph).

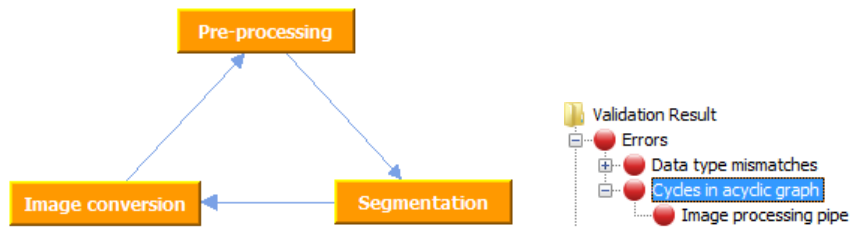


Figure 90 - Cycle detection (left: directed graph activity with cycle; right: validation error message)

In conclusion, the validation approach is effective. The designer can improve the workflow step-by-step, correcting problems and revalidating immediately (via a button at the bottom of the validation dialogue).

## 6.4 Summary

This chapter showed how the main components and functions of the system were tested and validated. The strategy thereby was to use realistic data and examples to evaluate individual aspects of the model and the implementation.

The semantic annotation of the items in the activity repository was used to evaluate the ontology for document image analysis with regards to completeness (coverage of the domain), using an independent source of knowledge (software method/tool descriptions and specifications) that was not used during the ontology engineering phase. The ontology proved to cover the relevant aspects of the target domain (small shortcomings were corrected).

Features that help users with workflow creation were tested using a basic document image processing pipeline. These mostly included semantics-supported functions such as filter-based search in workflow repositories, activity matching, and template concretisation, but

also other concepts such as user interaction and general workflow validation (to achieve completeness and validity).

This chapter comes close to confirming the PhD's hypothesis (semantic data and algorithmic use can assist users and simplify workflow design) and fulfilling objective 5. (test ontology and framework on real data) of Section 1.4. Nevertheless, a more comprehensive use case was deemed necessary for a final answer. The next chapter describes such use case, based on a real-world digitisation project.



## 7 Case Study: Digitisation of Historical Census Data

Chapter 6 described pragmatic experiments and basic use cases to work towards an answer whether the PhD's hypothesis is true or not. It was shown how workflow components can be semantically annotated using the proposed ontology for document image analysis. It was also shown how this information can be used (algorithmically) to simplify or assist with various workflow-related tasks. However, to better understand what users might be confronted with in a real scenario, larger and more complex workflows need to be considered. This chapter establishes an answer to the hypothesis (and objective 5., Section 1.4) by exploring problems and solutions for a UK-funded digitisation project.

Feasibility studies are often used to understand the problem characteristics, to determine the best processing approach (pipeline) using state-of-the-art methods, and ascertain the quality level possible regarding the digitisation of a large amount of physical material (books, newspapers etc.) of a given type or period. Designing a processing pipeline and evaluating it on sample data, however, is a complex task that requires experts with in-depth knowledge and a good overview of the domain of document image analysis. For this reason, digitisation is often outsourced to service providers which, in turn, frequently apply off-the-shelf text recognition systems (perhaps with some parameters set after a few experiments) combined with extensive manual correction.

Using a workflow system could be an alternative. Advantages include, but are not limited to: repeatable processes, reuse of existing solutions, improved understandability (and therefore easier validation of processes), inclusion and comparison of multiple analysis and recognition tools to achieve best results, and data provenance (e.g. to report back to funding bodies).

In this chapter:

- A real-world use case is presented: the Census 1961 digitisation feasibility study.
- The different aspects of the proposed workflow system are applied to the census scenario and the consequences are evaluated and discussed in depth.
- General conclusions from using the workflow-based approach and semantic enrichment are drawn.

## **Pipeline vs. Workflow**

In digitisation projects the term *pipeline* is often used to denote a collection of software tools and scripts that, in concert, take scanned documents as input, analyse and recognise the content, and output the results in the desired format. This resembles a workflow and one could argue that a pipeline *is* a workflow (albeit not formalised in a workflow language / model). Nevertheless, to avoid misunderstandings, the term *pipeline* will be used when referring to a process that was created and used for the Census 1961 Project. The term *workflow* will be used for something that was created with the proposed workflow system.

## **7.1 Census 1961 Digitisation Feasibility Study**

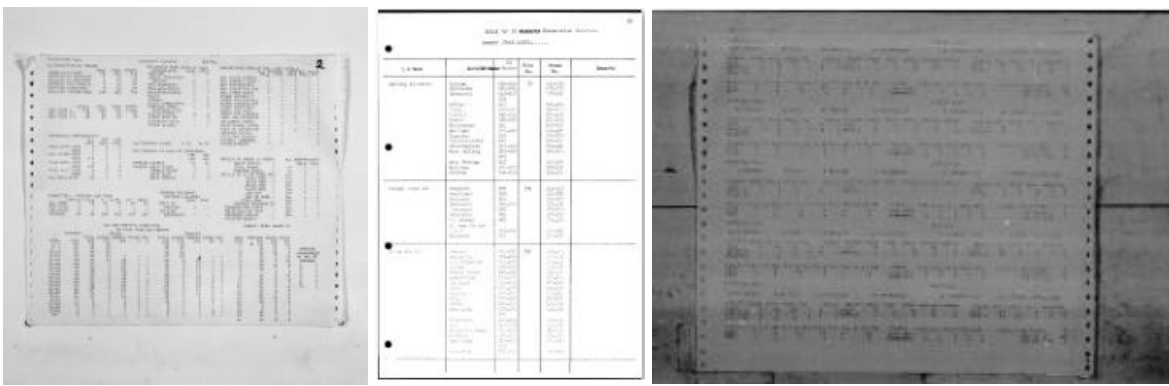
The study was conducted in two parts by the University of Salford in cooperation with the Office for National Statistics (ONS) [103] and Jisc [104] from September 2015 to December 2016 (see [105] and [106]). Its purpose was to ascertain the feasibility of digitising the 1961 Census data (from the only surviving set of low-quality scans of the original documents) in an automated way with the ultimate goal of establishing an online resource similar to recent censuses. The main questions to be answered were: What is the best way (selection of methods and parameters) of digitising the material to maximise the quality of the output and will the quality of the resulting information be high enough to satisfy the requirements of a trustworthy Census 1961 database for public access?

A prototype of a fully-functional digitisation pipeline was developed, including: image pre-processing, page analysis and recognition, post-processing, and data export. Each individual part of the pipeline was evaluated individually by testing a range of different approaches on a representative data sample. Well-established performance evaluation metrics were used to precisely measure the impact of variations in the workflow on different types of data (image quality, page content etc.).

Details about the census data, digitisation pipeline, and performance evaluation are provided next.

### 7.1.1 Dataset

The full 1961 Census data for England and Wales consists of approximately 140,000 scanned pages. From these, a representative subset of about 4,000 pages was selected. Most of the material consists of different types of tables that were either typeset (accumulative reports) or computer printouts (Small Area Statistics – SAS). The scans are characterised by a wide range of image quality with various production and scanning related issues and artefacts. Figure 91 shows three examples.



*Figure 91 - Examples of Census 1961 scans*

### 7.1.2 Ground Truth

Ground truth can be seen as the ideal result of a page recognition method, or, in other words, the result a perfect OCR system would produce. It is required as reference when evaluating OCR results. For the study, 41 representative images were chosen. The production was carried out with the Aletheia Document Analysis System [107] (see Figure 92). Where useful, pre-produced data (OCR results) from FineReader Engine 11 [108] was corrected, otherwise the ground truth was created from scratch. Both page layout and text content were transcribed. The output format is PAGE XML [88], a well-established data format representing physical and logical document page content.

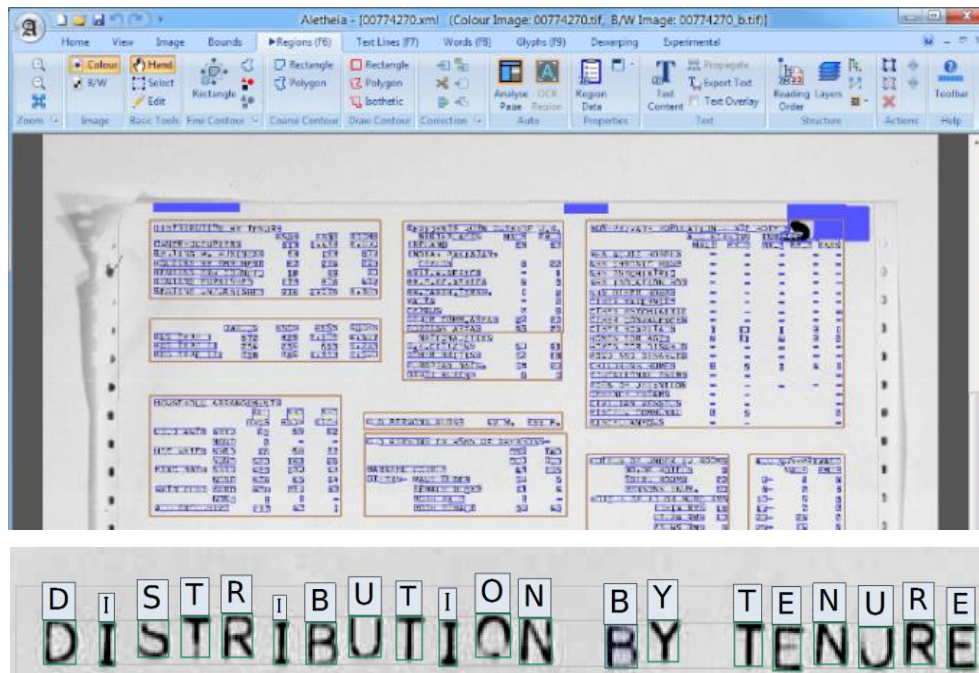


Figure 92 - Aletheia Document Analysis System

### 7.1.3 Digitisation Pipeline

As part of the study, a processing pipeline was designed and all essential parts were implemented and applied to the aforementioned dataset. Figure 93 shows an overview of the digitisation pipeline. The target was to extract the table information from image files (scans) and export it as comma-separated values (CSV) that can be fed into a database.

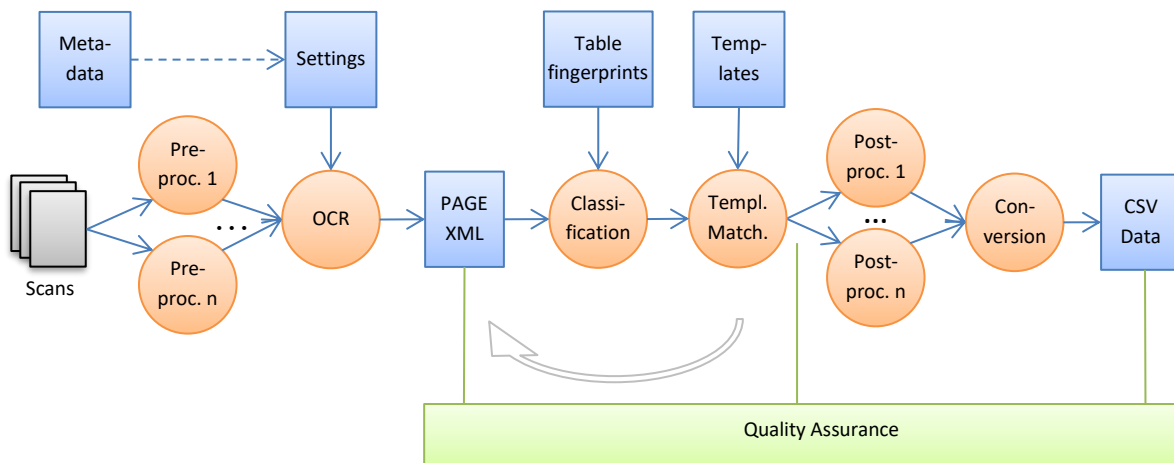


Figure 93 – Census digitisation pipeline

The processing steps can be summarised as follows:

- *Pre-processing*: Different pre-processing approaches tailored to different types of images. Main goal: improve OCR.
- *Page analysis and recognition*: Includes page segmentation, region classification, and text recognition. An *OCR engine* is used, and detailed, versatile outputs are produced (PAGE XML format). The engine is interchangeable, ABBYY FineReader Engine 11 [108] and the open source Tesseract OCR 3.04 [109] were used. Figure 94 illustrates the output of FineReader Engine 11 for an example from the Census data.
- *Table classification*: Identification of tables on image. Required for next step.
- *Template matching*: Alignment of table templates with OCR engine outputs (see Figure 95). Once aligned, the recognised text is transferred into the template.
- *Post-processing*: Improving results using rules.
- *Data conversion*: Creating comma-separated table from collection of PAGE XML files.

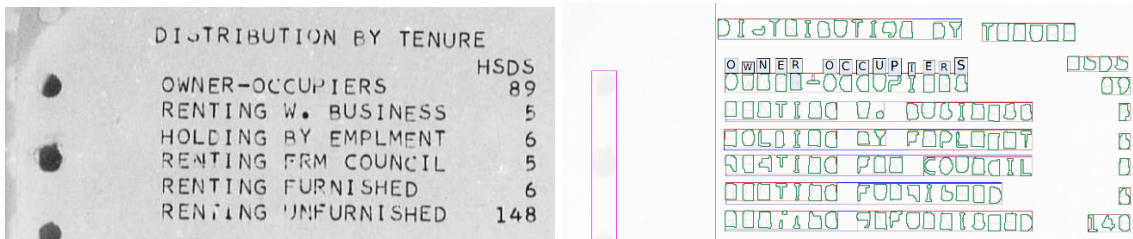


Figure 94 - Input image (left) and FineReader OCR result (right; visualised with Aletheia software tool)

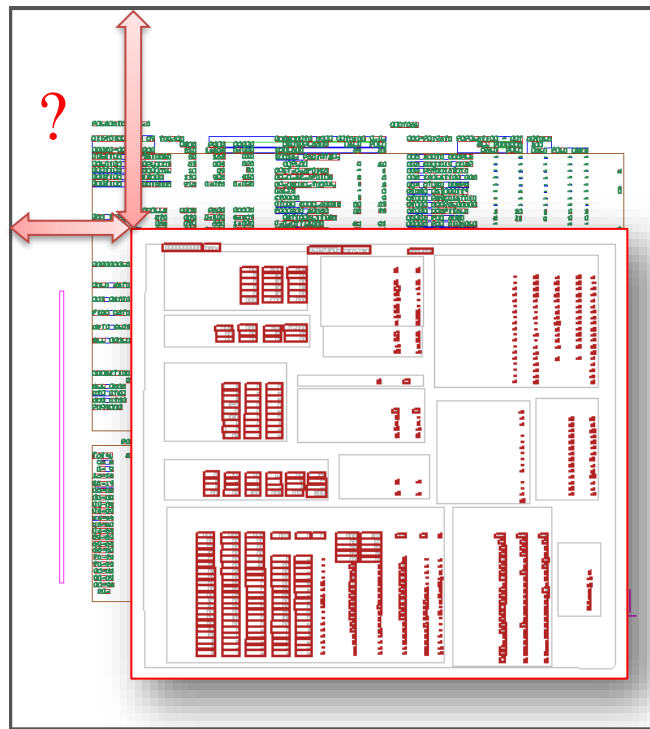


Figure 95 - Illustration of template matching (the goal is to find the best matching position of the template (red) with the page content (green))

#### 7.1.4 Performance Evaluation

The output of OCR engines can be evaluated by comparing it against the ground truth. A requirement is that both pieces of data (OCR result and ground truth) are available in the same data format.

Text-based performance measures were used to establish a quality baseline for both deployed OCR engines. To be able to consider a variety of pipeline setups efficiently, a

framework of scripts, evaluation tools, and analysis approaches was created. That way, experiments could be set up and modified easily with as little manual labour as possible. Figure 96 provides an overview of the framework. A cascade of scripts processes a subset of the Census data using all possible combinations of pre-processing steps, OCR engine settings, evaluation tools, and evaluation settings.

For the most part, the Layout Evaluation Tool [110], developed by the PRImA Research Lab, was used. Figure 97 shows the graphical interface of the tool, visualising an evaluation result.

Based on the execution and experience of the Census 1961 feasibility study, the next section discusses how the workflow system with semantic features can aid with this project and other projects of this nature.

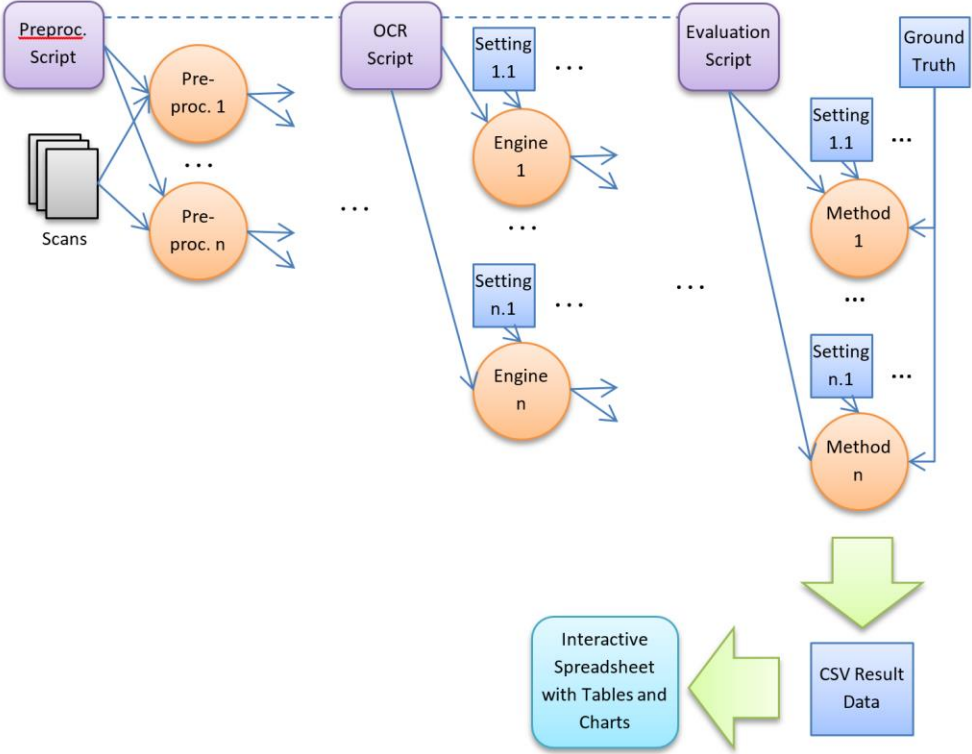


Figure 96 - Processing and evaluation framework

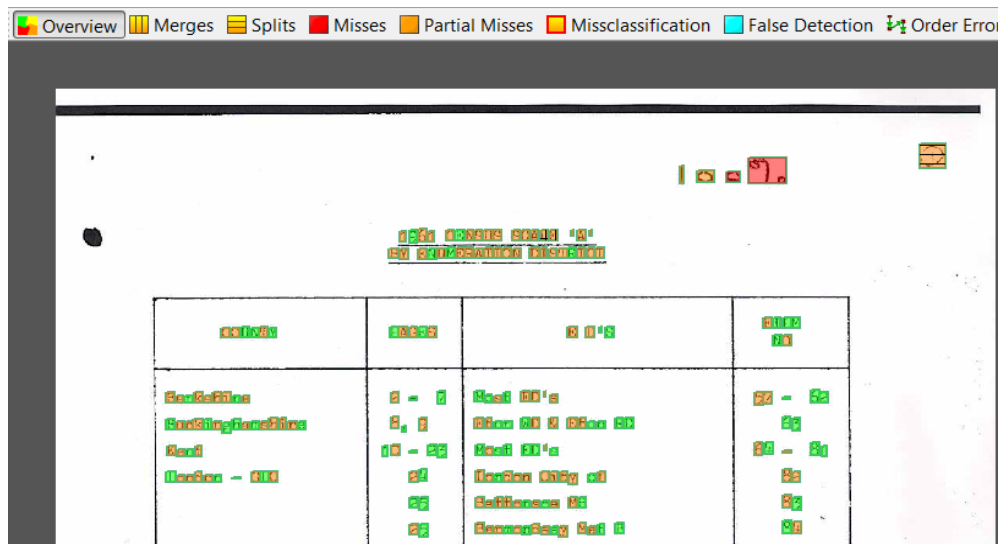


Figure 97 - Visualisation of evaluation result within the PRIMA Layout Evaluation tool

## 7.2 General Considerations for Applying the Workflow System

Using the Census 1961 project as a representative for feasibility studies, someone tasked with carrying out a pilot for a similar digitisation project will be confronted with the following questions:

- What pre-processing steps are suitable for scans of printed documents with issues caused by aging?
- What recognition engines are available that fit our needs (licence, execution platform etc.)
- Can the data be processed directly or is conversion required and if yes, what are suitable converters?
- What does a typical processing pipeline look like?
- If additional data is required (e.g. ground truth), how can it be produced?
- How can the performance of a pipeline be measured?

Depending on the experience of the person or team conducting the project, it might not be obvious where to start. A reasonable first step would be to search for existing solutions. In the context of workflows, template repositories represent knowledge and information that



experts provided and made available in form of proven methodologies. In the next two sections, the creation of workflow templates and their usage will be described (from two different points of view).

### 7.3 Workflow Template Creation

Experts may want to create workflow templates for either their own future use or for sharing knowledge. Similar to object-oriented software design, reusability can be improved by using a modular design. In terms of workflows this means, instead of creating one large template of a digitisation pipeline, the expert can create several more generic sub-workflows which model specific parts of the pipeline.

In the following, workflow templates extrapolated from the census study are presented.

#### 7.3.1 Form / Table Analysis and Recognition

Figure 98 shows the UML notation of a workflow template for table recognition (also applicable to form recognition). It consists of a Directed Graph activity and four abstract atomic activities. Table 11 shows the semantic labels that were assigned to the main activity of the template and its data ports. Data types were not specified to make the template more generic.

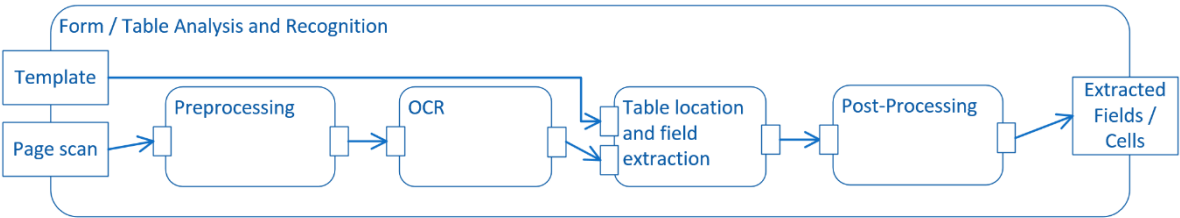


Figure 98 - Workflow template for form or table recognition (UML notation)

Table 11 - Semantic labels of form or table analysis and recognition activity

Activity / Port	Label Group	Semantic Label
<b>Form / table analysis and recognition</b>	Data creation / transformation	Information Extraction
	Processing level	Classification / recognition
	Adaptability	Configurable
	Activity domain	Table / form analysis and recognition
Image	Data granularity	Page
	Content encoding	Raster image
	Content type	Data
	Content of interest	Text
		Tables / forms
	Original source	Paper document
Table template	Data granularity	Cell
		Glyph
	Precision	Ground truth
	Content encoding	Structured
	Content type	Data
		Location
	Content of interest	Text
		Tables / forms
Extracted fields	Data granularity	Cell
	Precision	Estimated
	Content encoding	Structured
	Content type	Data
		Annotations
	Content of interest	Text
	Tables / forms	

### 7.3.2 Pre-Processing Method Selection

Different image pre-processing methods were applied within the census project, depending on which data subset an image belongs to. This aspect is represented by the workflow depicted in Figure 99. Therein, one of two pre-processing tools is called with specific (external) settings or no pre-processing is performed. The workflow is not a template since it uses concrete activities, but it can be used as a module that can be inserted into different other workflows.

A second workflow was created to model the methodology of selecting a suitable image pre-processing method (see Figure 100 for UML notation). It runs different pre-processing methods for all input images, runs an image analysis and recognition method, evaluates the results, and outputs the best pre-processing method.

The module for conditional pre-processing was inserted using the “Search repository for activity” feature of the editor. Alternatively, it could be added as a child workflow, but this functionality is not fully implemented.

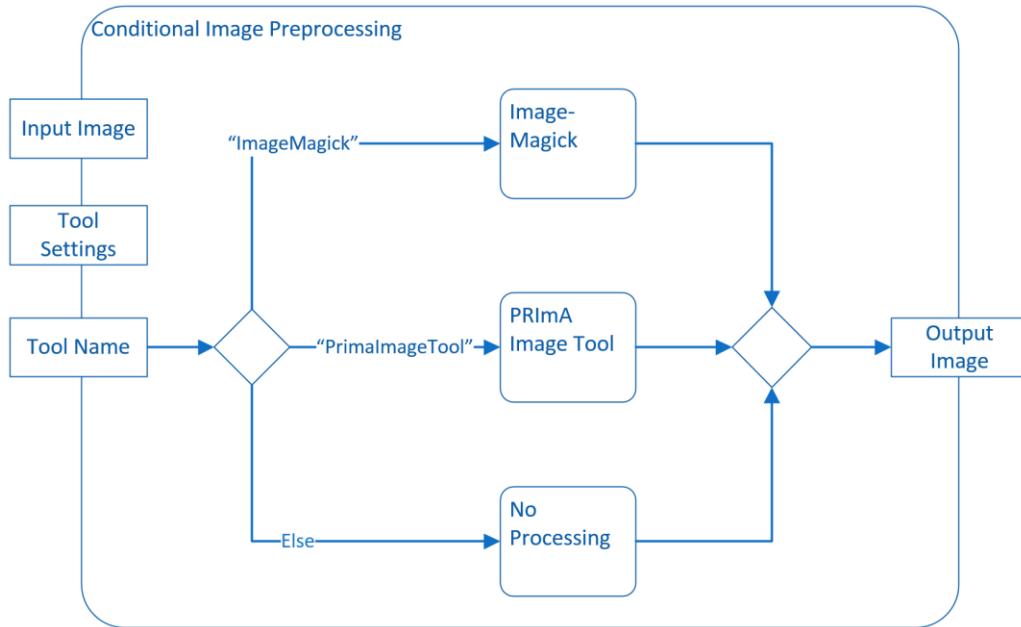


Figure 99 - Workflow for conditional image pre-processing (UML notation)

The workflow has two abstract atomic activities (“Analysis / Recognition” and “Evaluation”) and is therefore a template that must be concretised before it can be executed. The rationale for this is to allow the selection of pre-processing methods for different use cases, depending on what the images are used for.

Pre-processing tools and settings are provided via a data table. In the UML diagram the table is illustrated like an activity with output ports only. Figure 101 shows the template and the content of the data table in the Workflow Editor.

Within the census project, more pre-processing steps were evaluated by combining different methods. For simplicity, this was not modelled here. Adding more branches to the conditional pre-processing module could be used to this end.

Table 12 shows selected semantic labels of the template. Most important are the labels for the root activity and the abstract activities since these are the objects a user is most likely to

interact with. The other activities are part of the inner workings of the workflow and will be handled as a black box by most users.

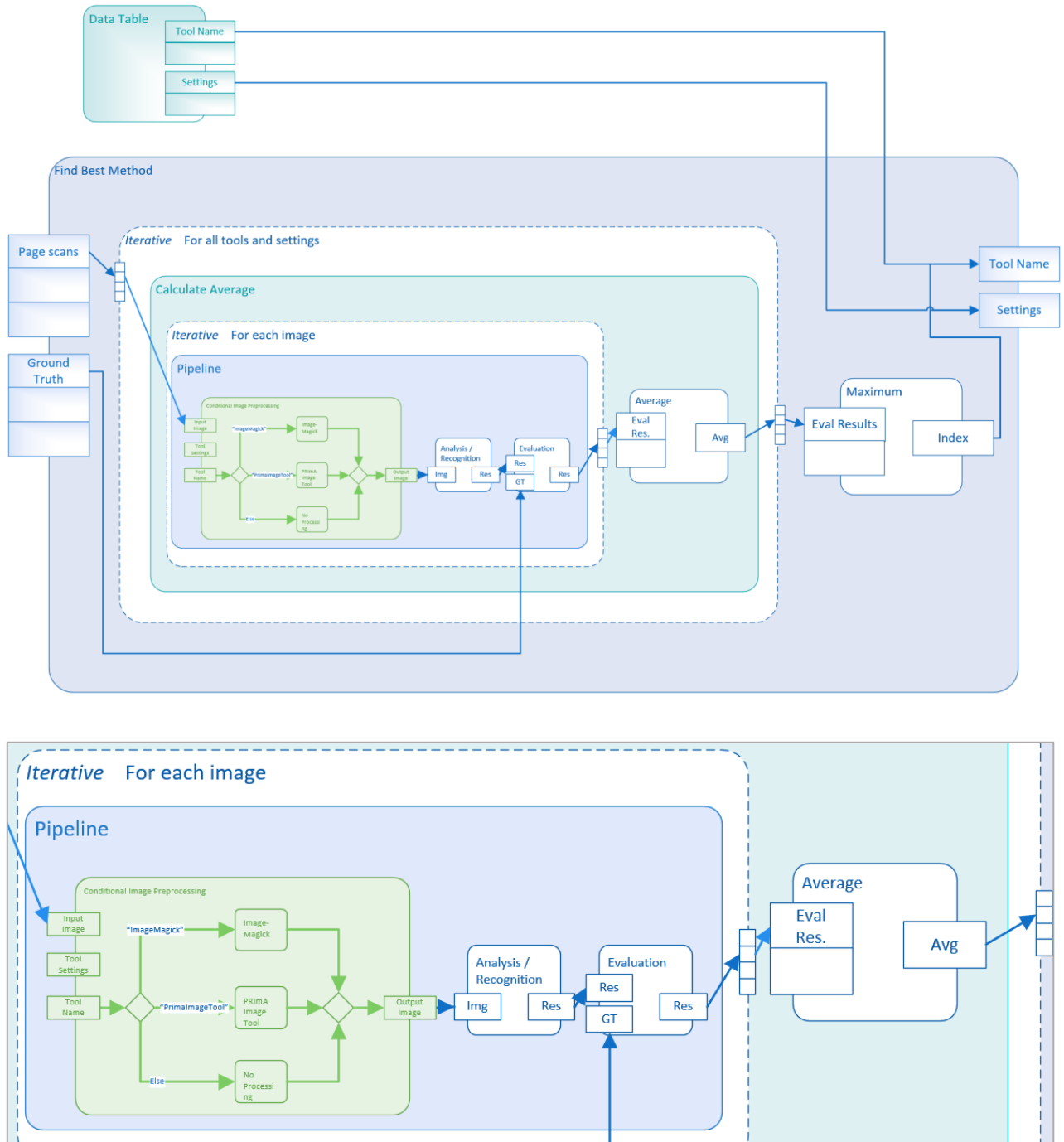


Figure 100 - Workflow template for image pre-processing selection; top: overview, bottom: enlarged inner for loop (UML, simplified)

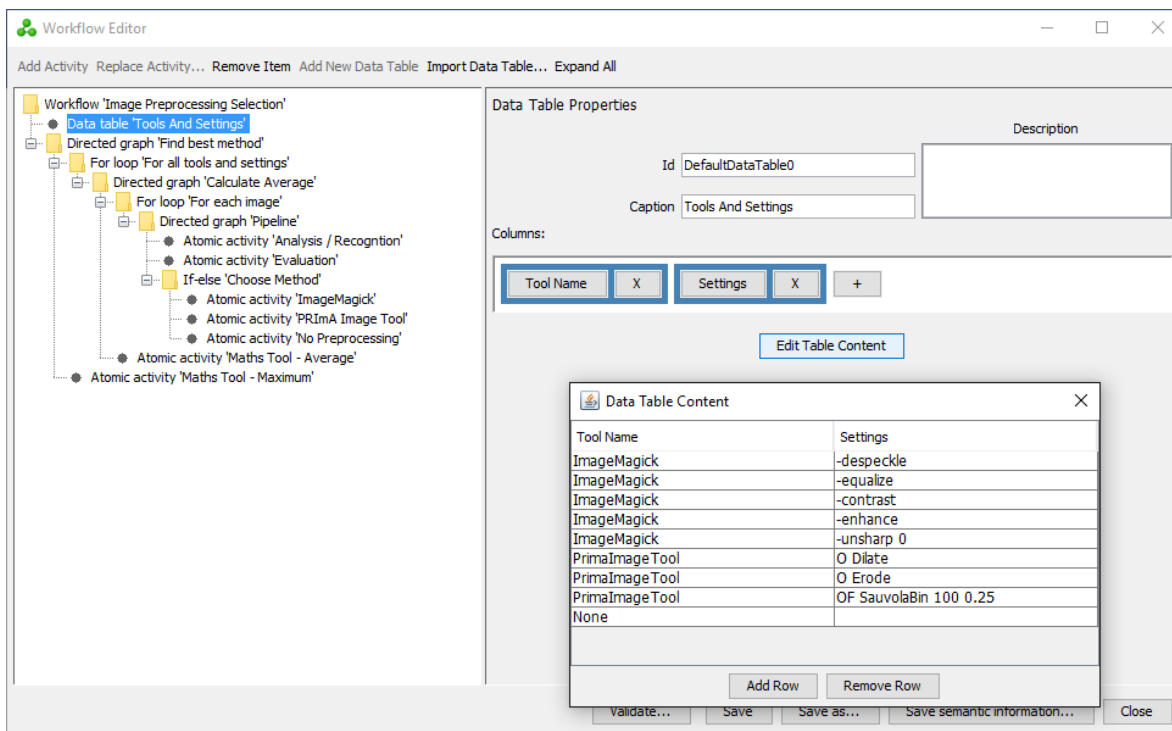


Figure 101 - Image pre-processing method selection template in Workflow Editor

Table 12 - Selected semantic labels for Image Pre-Processing Selection workflow template

Activity / Port	Label Group	Semantic Label
<b>Root activity</b>	Data creation	Information extraction
	Automation	Automated
	Adaptability	Configurable
	Activity domain	Comparative performance analysis
<b>Analysis / Recognition</b>	Data creation	Information extraction
	Automation	Automated
	Processing level	High-level
	Activity Domain	Content analysis and recognition
<b>Evaluation</b>	Data creation	Information extraction
	Automation	Automated
	Activity domain	Comparative performance analysis
Port "Recognition Result"	Precision	Estimated
Port "Ground Truth"	Precision	Ground truth
Port "Evaluation Result"	Precision	Measured
	Content type	Performance information

### 7.3.3 Table Template Creation

An example that a workflow can be useful even if it cannot be executed in a fully automated way is provided in Figure 102. For the census digitisation pipeline to work, table templates are required. They are used to align the results of an OCR engine with a given template (containing the layout and cells of one or multiple tables that appear on the respective page of the census material). The activity “Correction” (highlighted) is labelled as “Automation – Manual”, indicating that human intervention is required. In theory, this can still be automated by adopting crowdsourcing where the correction task is presented to volunteers or paid workers. Similar hybrid workflows are already used in the industry.

The next section focusses on workflow template discovery.

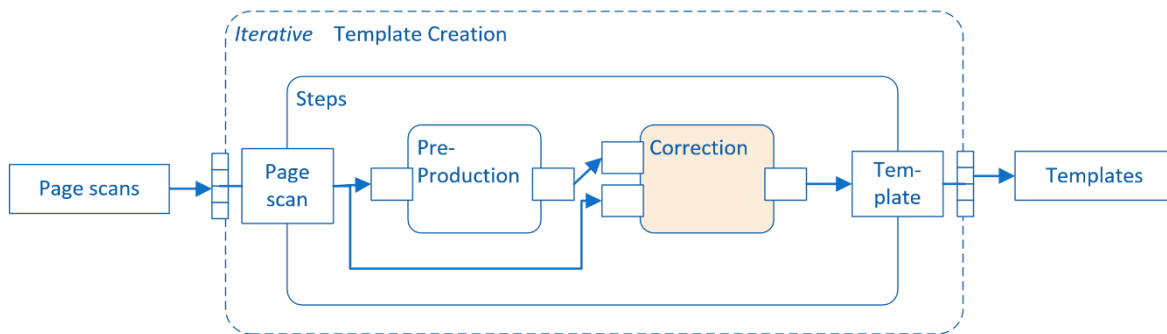


Figure 102 - Table template creation workflow (UML)

## 7.4 Finding Workflow Templates

In this section, it will be described how suitable workflow templates can be discovered. Two different angles are explored, referred to as *User L* and *User C*. Neither user is an expert in document image analysis. Let User L be from a library or digital archive background and let User C be from a computer science / software development background. Other user groups exist but two examples are thought to be sufficient to demonstrate the flexibility of the proposed system.

## Workflow Discovery - User L:

User L would be an expert with regard to the census data, having detailed knowledge on the condition, metadata, formats etc. The first step is to select a suitable workflow repository within the Repository Hub tool. In the future, repositories could be organised by domain, if the sizes become difficult to handle. For this experiment, a combined repository was created that contains all activities and workflow templates (as introduced earlier). Since the repository contains all workflow sources, the step of selecting a repository is trivial.

Searching for workflows which take tables as input data (Content of interest – Visual – Mixed – Tables / Forms) results in two items (see Figure 103):

- ABBYY FineReader Engine and
- Form or table analysis and recognition.

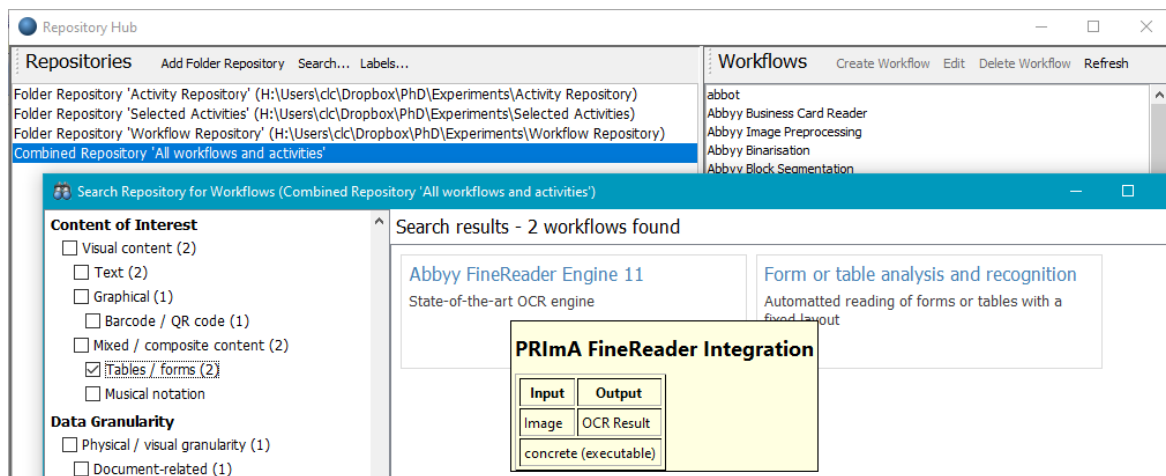


Figure 103 - Workflow discovery by content of interest

If too many items are returned, the list can be narrowed down using other filters (e.g. execution environment and licence).

When looking at the search results, FineReader engine seems the better choice at first since it requires only an image whereas the table analysis workflow requires a table template as well. Furthermore, FineReader is a concrete workflow and therefore directly executable. But, investigating further, it becomes clear that FineReader is too generic for the task at hand (extracting table data to eventually feed it into a database). FineReader outputs an OCR

result, a PAGE XML file, in this instance. The librarian might already know or can find out that the PAGE format does not support a table structure and the extracted data is therefore not suitable. In a production system, concrete workflows could be tested with example data in case not enough information is provided within the workflow documentation.

The output of the table analysis workflow is called (more promisingly) “Extracted Fields” and it is selected as starting point (continued in next subsection).

### Workflow Discovery - User C:

It is assumed that a computer scientist understands that somewhere in the process the text of the table cells needs to be recognised. The repository can be searched for workflows that perform Optical Character Recognition (OCR) (Activity domain – Computing – Visual computing – Content analysis – OCR). By default, only the root activities of workflows are regarded when filtering. To include all activities of all workflows, the search option “Include child activities” can be selected. This returns all workflows or workflow templates that contain an OCR step anywhere in their structure. For the example repository this returns nine result items. Further refinement using “Automation – Automated” and “Adaptability – Configurable” narrows the list down to four items (see Figure 104).

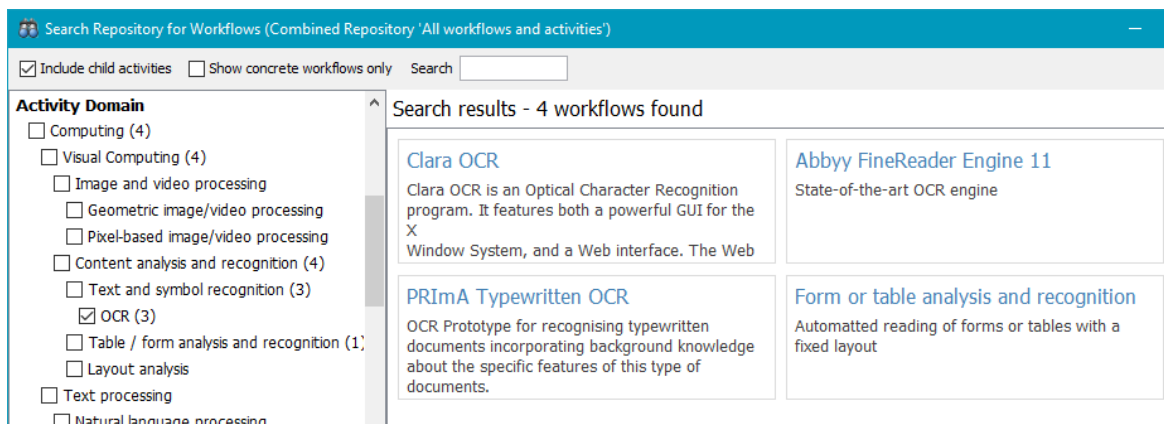


Figure 104 - Workflow discovery by activity domain

The table recognition workflow is amongst the result items and the user could read the descriptions to make a decision on which workflow to use. A useful feature of the filter



controls (the checkboxes) is that they can be used to see what other semantic labels the current selection of workflows contains. This can help to find more suitable filter options, if required.

Having selected a workflow or template, the steps required to arrive at an executable workflow are described in the following sections.

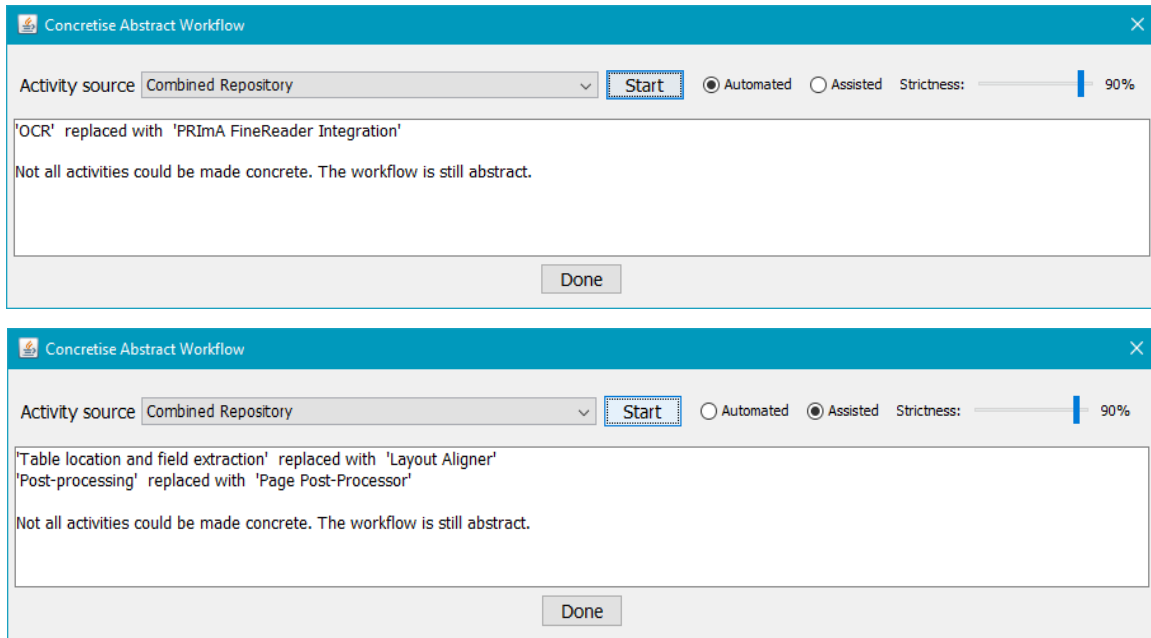
## 7.5 Template Concretisation

This section explains how to continue creating an executable workflow from the initially selected template. As before, the two viewpoints are used.

### **Concretisation – User L:**

A novice to workflow systems will not be aware of all available routes to a successful workflow. But since a suitable template was found in the previous step, it is straightforward to use that as a basis and open the workflow in the editor. The Workflow Editor displays that the workflow is abstract and not executable. The concretisation is offered as a solution (see subsection 5.4.8) and can be carried out with the combined repository (all workflows and activities) in the automated mode.

For the example, the concretisation replaced one abstract activity (OCR replaced with PRImA FineReader Integration), but the remaining abstract activities could not be replaced using the default settings (strictness 90%). Figure 105 shows the initial result.



*Figure 105 - Partial concretisation of table recognition template (top: result of automated process; bottom: result of assisted process)*

The user now has two options on how to continue (in future versions of the prototype, the dialogue could inform the user of those options):

- (1) Gradually decrease the strictness threshold and re-concretise until all activities are replaced or the lowest threshold is reached. For the template at hand this strictness is 80%. Nevertheless, this can lead to unsuitable workflows (see subsection 6.3.3) and an inexperienced user might not feel confident to change advanced settings. Instead, option two can be used.
- (2) Use the assisted concretisation. By switching from “Automated” to “Assisted”, conflicts can be resolved by the user interactively. For the example workflow template, this results in three interactions:
  - “Pre-processing” activity: Twelve matching items are found, none of which are obvious choices for the table recognition workflow. In such a case, the user can opt to ignore this abstract activity (“Do not replace”) and handle the problem later.

- “Table location and field extraction”: Five activities are shown as matches (sorted by match score). The first two (“Layout Aligner” and “Table Exporter”) are related to table recognition and the user can choose one by clicking on the result item. It is assumed that the best match was selected (problems caused by wrong choices in this step can be uncovered and corrected later using workflow validation).
- “Post-processing”: The user is presented with nine items, the best match being the “Page Post-Processor” (assumed to be selected for this use case).

The result of the concretisation is a workflow which is still partially abstract (the pre-processing step) and needs further refinement (see Figure 105 bottom), which is described in later subsections. Before that, an alternative route is explained in the following.

### **Concretisation – User C:**

Inspecting the workflow template for table recognition, it becomes apparent that the input is one single image. The census data, however, consists of whole sets of images. A user with experience in software development will know that this can be resolved using a loop of some kind. After starting a new workflow from scratch (“Create Workflow” in the Repository Hub), a loop can be added by creating a for-loop activity. An initial input port representing a data collection ensures that the workflow can handle the census dataset.

The previously found template “Form or table analysis and recognition” can be added to the loop by adding a child activity via the search dialogue (“Add activity” – “Search repository for activity”). A simple text search is sufficient since the name is known (see Figure 106).

Instead of concretising the workflow using the automated or assisted method that was used in the User L’s scenario, a manual approach can be used with more control over the activity replacement. The user thereby replaces all atomic activities that are marked as “abstract” with concrete ones from a repository (or a newly created one).

Table 13 shows the matching results for the four abstract activities of the table recognition template. As semantic filter “Automation – Automated” was used. Choosing a suitable

activity is the responsibility of the user. The decision-making process can include reading the descriptions of the best matching activities and possibly test them with the target data.

Replacing an abstract activity has an influence on the match scores of the remaining activities that are yet to be replaced. The order of replacing can therefore be of relevance. If possible, the user should start with the most important activity or the one they are most confident about. The two rightmost columns of Table 13 show the order in which the activities were replaced in this experiment and the new match scores (just before an activity is replaced and after other activities were replaced). The symbols “↑”, “↓”, “=” indicate whether the match scores increased, decreased, or stayed identical respectively.

The pre-processing activity presents the same problem as mentioned earlier (user L’s point of view). Even a domain expert cannot make an informed decision in cases like this. The next section focusses on possible solutions.

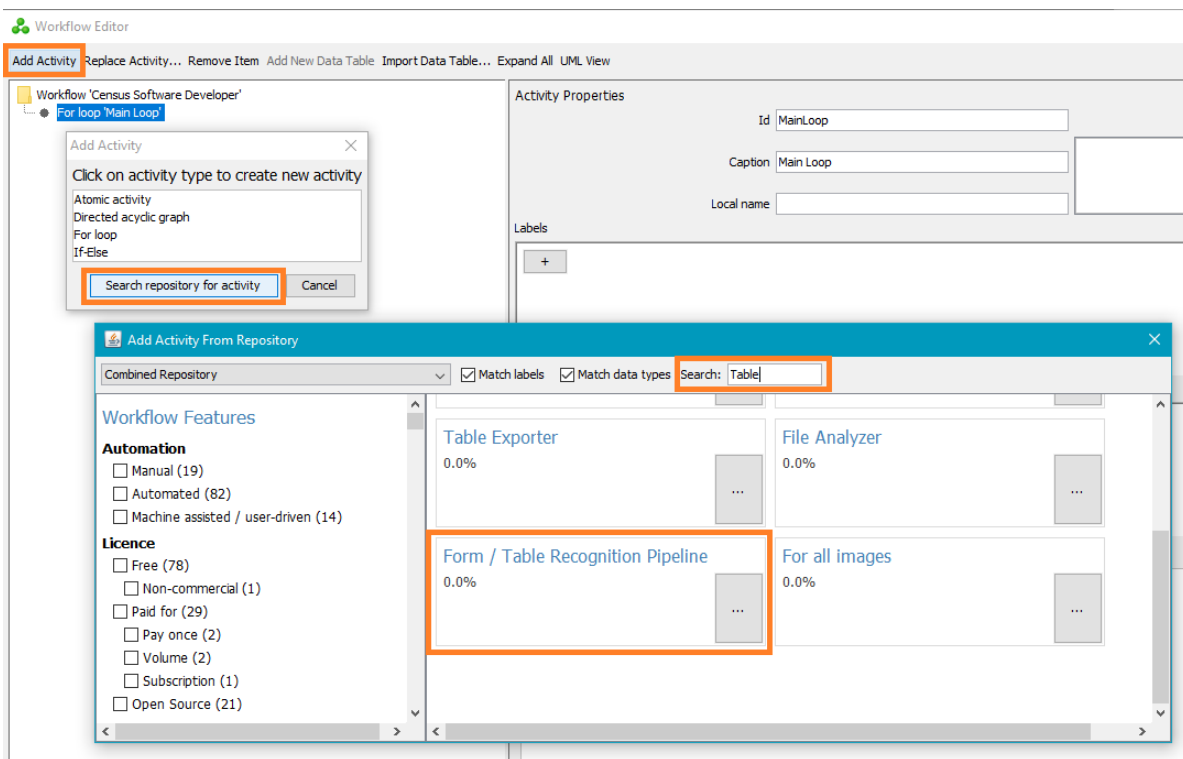


Figure 106 - Adding a new activity via text search

Table 13 - Matching results for table recognition template

<b>Abstract Atomic Activity</b>	<b>Matching Concrete Activities with Highest Scores</b>	<b>Initial Match Score</b>	<b>Replacement Order</b>	<b>New Match Score</b>
Pre-processing	Document Deskewer	83%	N/A (not replaced)	= 83%
	Hot Metal Font Enhancer	83%		= 83%
	Border Removal	83%		= 83%
	Hectography Foreground Extractor	78%		= 78%
	Page Curl Correction	78%		= 78%
	Dewarping	78%		= 78%
	ImageMagick	75%		= 75%
OCR	PRImA FineReader Integration	84%	1	N/A
	PRImA TypeWritten OCR	76%		
	Ocrad	73%		
	OCRopus	71%		
Table Location and Field Extraction	Layout Aligner	86%	2	↑ 88%
	Table Exporter	66%		↑ 72%
	Page post-processor	60%		↑ 66%
Post-Processing	Page post-processor	80%	3	↑ 84%
	Text and Error Profiler	76%		= 76%
	IMPACT Spelling variation tool	72%		↓ 66%
	PRImA Text Normalisation	71%		↓ 63%
	Unpaper	66%		↓ 55%
	Word Segmenter	49%		↑ 58%

## 7.6 Pre-Processing Activity Selection

It is the responsibility of the designer to provide as much information as possible for each component of a workflow template. This can include related templates, recommended software tools (in form of atomic activities), and parameter values including their meaning. For the pre-processing step of the example template, it is recommended to select a method depending on the input data, using the workflow for image pre-processing selection (see 7.3).

If there are insufficient resources for ground truth production (involves manual work, but required for the selection workflow), an alternative is to use a pre-processing method that improved the recognition results in most cases and never decreased the success (for instance ImageMagick with the “enhance” option). If the template designer does not include enough

information, the user of the workflow system can search the repository for suitable workflows from the domain of performance analysis. Suitable semantic search filters are:

- Activity domain – Computing – Performance evaluation
- Data creation / transformation – Enhancement (with option “Include child activities”)
- Input data – Content encoding – Raster image

The pre-processing selection template (see again Figure 99) can be discovered in such a way. The workflow contains concrete image processing tools, but it is generic regarding the analysis / recognition that is performed after the pre-processing and the evaluation of the results. Again, the template designer should inform the user which steps are required to arrive at an executable workflow. Alternatively, the steps from the previous subsection can be repeated here. In conclusion, the first step is to replace the “Analysis / Recognition” placeholder with the same OCR method that was used earlier (ABBYY FineReader).

The second step is to find an appropriate performance evaluation method. Since the OCR step is already integrated, a search by activity matching returns several valid candidates, including:

- PRImA OCR Evaluation.
- NCSR OCR Evaluation.
- HOCR Eval.

The decision can then be based on secondary semantic properties such as processing platform and licence. For this case study, the PRImA OCR Evaluation is selected.

To be able to execute a workflow, in most cases, it also needs to be connected to an external data source. The next section explains how this can be done for the pre-processing selection workflow.

## 7.7 Providing External Input Data

A complete workflow system will offer assistive features to import external data and annotate it semantically. The prototype does not contain such functionality, but it can be emulated using data tables (see Section 4.1).

The workflow editor offers assistance to reach an executable workflow (validation dialogue; see 5.4.7). With regard to input data, unconnected ports are listed and solutions are presented (Figure 107). One data table with the following two columns is enough:

- 1) Census pages
- 2) Corresponding ground truth

The table data can then be routed to the respective input ports (displayed in the validation panel). What has been disregarded in the described procedure so far is where the actual data (files, for instance) originates from. The census pages are available as images and can be linked directly. A production workflow system is likely to have several mechanisms to do so, such as local files, database connections etc. The ground truth data, however, needs to be created manually. Two possible scenarios are described below.

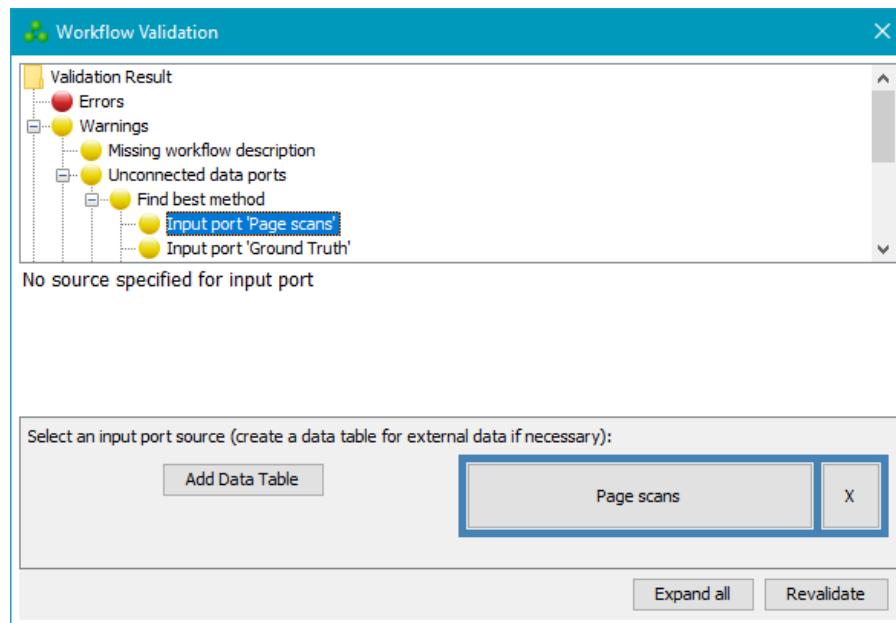


Figure 107 - Unconnected input ports in pre-processing selection workflow

### **Ground Truth Creation – User L:**

Ground truth data for the census files might already be available within the library or the user has a favourite tool to create ground truth. If the data format does not match the required format (or formats) of the evaluation activity from the workflow (i.e. the PRImA OCR Evaluation tool), the workflow is faulty. The validation reports this as an error and offers three solutions:

- Check the data types and correct them if they are wrong.
- Use another activity that has suitable data ports.
- Add a converter.

The latter option was discussed in Section 6.3 and is the solution that needs the least knowledge regarding the workflow design. Nevertheless, a converter can only be added if a suitable one is listed in an activity repository, otherwise the operation ends with a warning and only an abstract converter activity is added. That, in return, means that the workflow is not executable until a concrete converter is provided or the workflow is changed according to the remaining solutions listed above.

If, for this example, the ground truth is available in ALTO XML format (widely used by libraries), a suitable converter exists: the PRImA Page Conversion Tool. It can convert different page layout formats (including ALTO) to PAGE XML, which is an allowed data type for the evaluation activity.

Figure 108 shows the result of the modifications made to the workflow template, with added data table, filled in concrete atomic activities, and data conversion step.



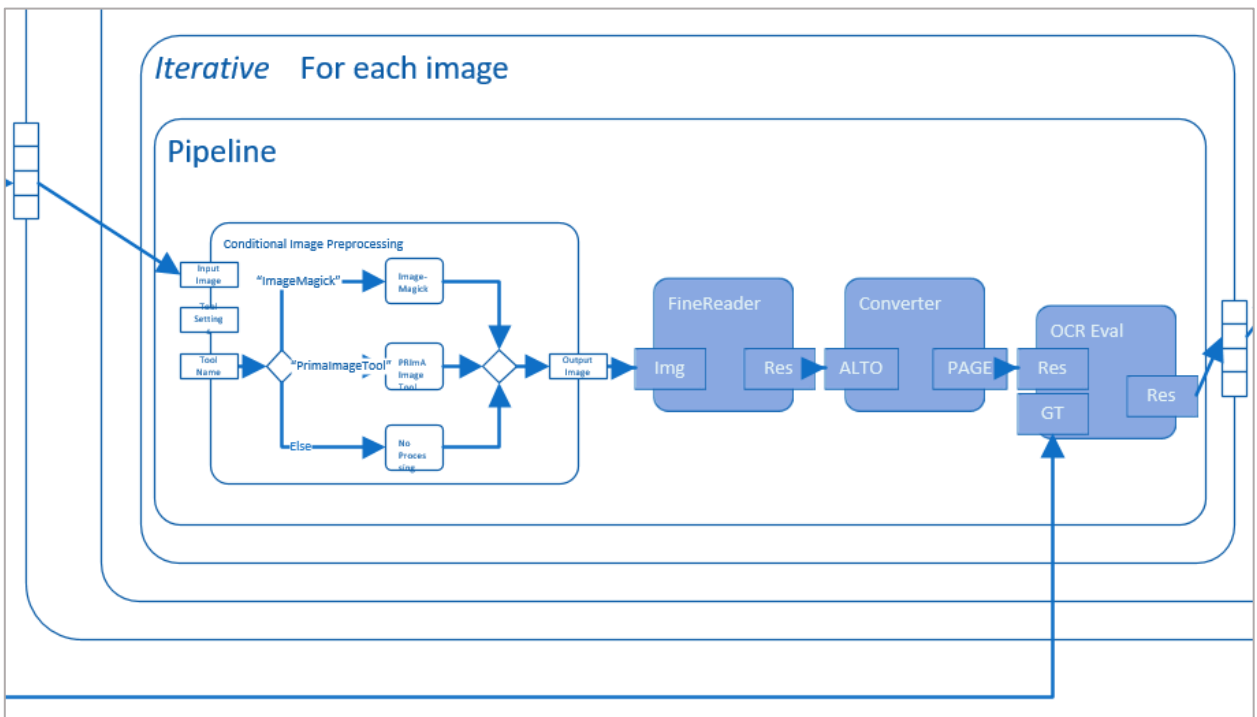
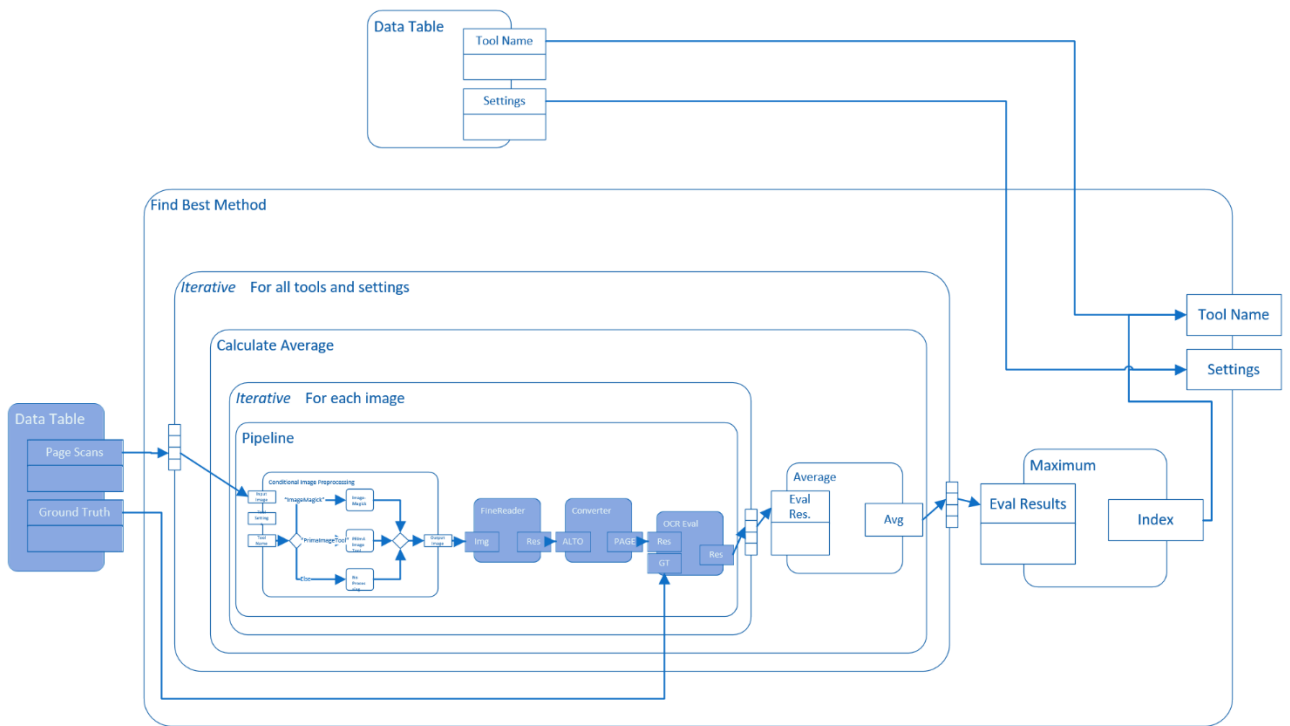


Figure 108 - Executable image pre-processing selection workflow; top: whole workflow, bottom: enlarged inner for loop (changes made to template highlighted)

### **Ground Truth Creation – User C:**

If ground truth data needs to be created from scratch and no specific ground truthing tool is to be used, the user can search for suitable workflows or activities in the respective repositories. The OCR Evaluation Tool allows for inputs in form of plain text files or PAGE XML files. The user can run a semantic search, looking for workflows with the following labels:

- Automation – Manual or Automation – Assisted
- Output data – Ground truth / gold standard
- Content of interest – Visual content – Text

One matching tool is the Aletheia Document Analysis System. Once the ground truth is created with Aletheia, the data can be used straightaway, without the need for a converter.

The output of the image pre-processing selection workflow is a tool ID and corresponding settings that can be used in the census digitisation pipeline to improve the overall result quality. How the pre-processing step can be integrated in the census workflow is explained in the next section.

## **7.8 Adding a Pre-Processing Step**

The census template concretisation (Section 7.5) is incomplete, with the pre-processing step still being abstract. The previous subsections described how a suitable image processing method can be selected. This method now needs to be integrated into the digitisation workflow template. Again, two possible paths are considered.

### **Adding an Activity – User L:**

The most direct way of adding the selected pre-processing activity is to use the “Replace Activity” feature of the Workflow Editor and search for the software tool in the activity repository (using the tool ID that was an output value of the selection workflow). If the pre-

processing tool is not listed in the repository, it can be imported directly from the image pre-processing template, using following steps:

- Select the abstract “Pre-processing” activity.
- Choose “Replace Activity” in the workflow editor.
- Select “Activities of a selected workflow...” as source.
- Find and select the image pre-processing template.
- Choose the desired activity (e.g. ImageMagick tool).
- Finish the replacement operation as usual (see Section 5.4.6).

The tool settings, also an output of the selection workflow, can be copied directly to the “fixed value” field of the respective input port.

### **Adding an Activity – User C:**

Although the steps described above are valid and produce an executable workflow, the result is not future-proof since the pre-processing activity would have to be changed each time the input data changes (a different set of census data, for instance). Instead, a more generic solution can be used. User C might have noticed (or the template designer added a note) that a section of the pre-processing selection template is a *workflow module*, available as separate workflow in the repository (called “Conditional Image Pre-processing”, see again 7.3).

The abstract pre-processing activity of the census digitisation workflow can be replaced by the discovered sub-workflow. The input values for tool name and settings can be copied from the result of the executed selection workflow. If the census input data changes, the pre-processing selection can simply be run again and the result values can be copied to the input ports of the conditional image pre-processing activity. There is no need to change the workflow structure.

The main steps towards a usable workflow have been taken. The next section describes the remaining tasks.

## 7.9 Finalising the Census Workflow

The remainder of the work required to reach an executable and effective workflow is an iterative process of workflow validation and handling of issues that arise. In addition to problems that were discussed earlier, this could include the following items (organised by the two scenarios).

### Workflow Finalisation – User L:

The workflow template for table recognition was designed to process only one input image. If the user connects a data collection (for instance a column of a data table) with the input port of the root activity, a mismatch is produced and the validation reports this as error (see Figure 109).

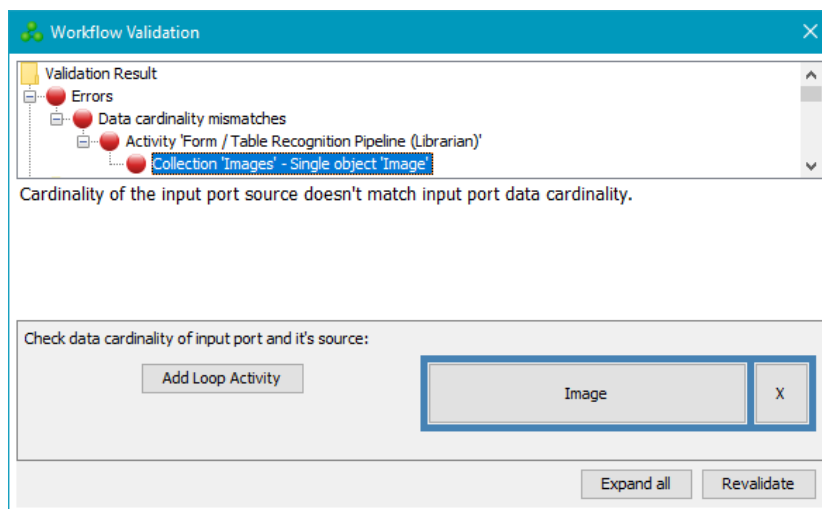


Figure 109 - Validation error for data cardinality mismatch

At this point, the user must intervene and make a decision (given it was not just a mistake). The workflow system cannot know whether the intention is to execute the corresponding section of the workflow only once with a specific data item from the source collection or whether to iterate over the data collection and repeat the processing for each item.

In the census example, there is a dataset consisting of multiple page scans and a digitisation pipeline that deals with one page at a time. Therefore, the correct solution is to

wrap the pipeline in a for-loop activity. Figure 110 shows the transformations the workflow systems applies to the workflow.

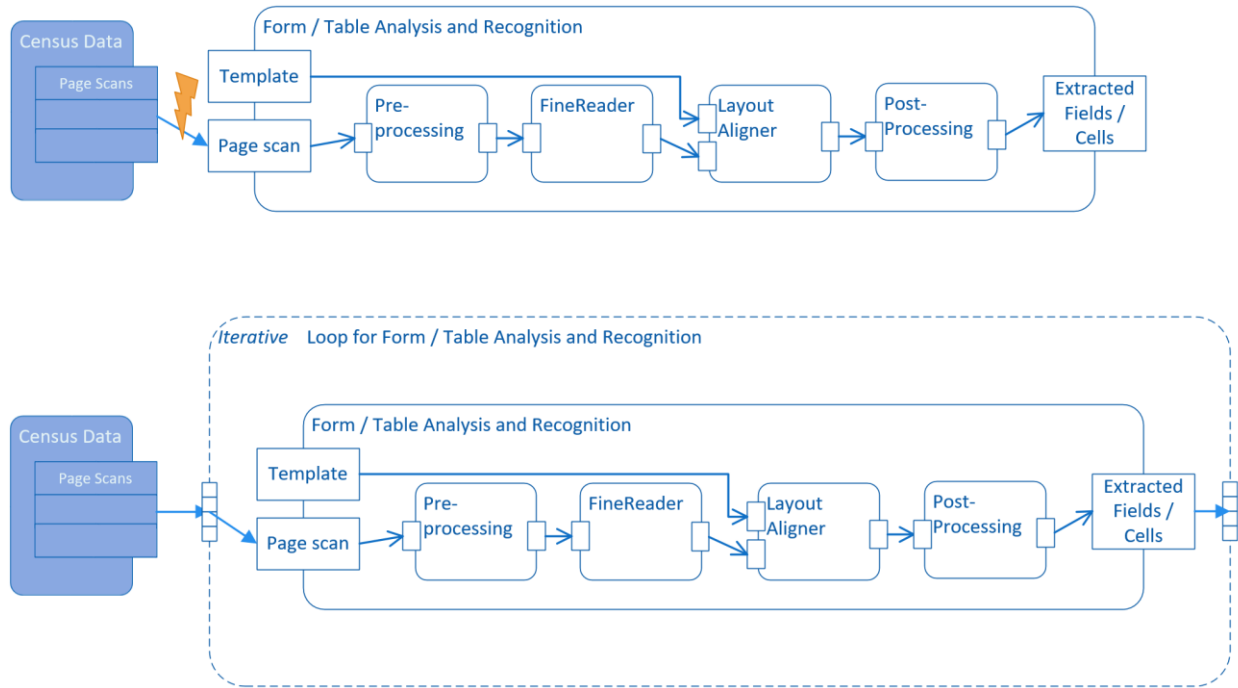


Figure 110 - Adding a loop to resolve data cardinality mismatch (top: cardinality mismatch; bottom: solution using a for-loop activity)

### Workflow Finalisation – User C:

As mentioned in Section 7.5, in this scenario, the user has already created a loop activity and added the digitisation pipeline as child activity. Nevertheless, the so-called loop ports also need to be connected to the correct ports in order to indicate which item from a data collection to use at any given time and how many iterations are to be executed. The workflow validation warns of unconnected ports, but the actual “wiring” is the responsibility of the user. This can be achieved by providing *sources* for input ports and *forwarding* for output ports (see 5.4.1).

A few remaining points to complete the workflow are only mentioned in short since they can be dealt with in similar fashion as other issues described before:

- Template data: The pipeline requires an input called “Template”. A suitable workflow was provided by the template designer (“Table Template Creation”) and can be used.
- Table classification: The census data contains different types of tables, each of which requires a different table template (see previous point). An additional activity needs to be used in case different table types are to be fed into the table recognition workflow. A suitable atomic activity exists (“PRImA Table Classifier”) and can be integrated into the workflow.

## 7.10 Discussion

The aim of this case study was to investigate whether the proposed semantics-enabled workflow system can be used to replicate a pilot project like the Census 1961 Feasibility Study. The focus thereby were the semantic features and their impact on assistive or automated functions of the system.

Table 14 provides an overview of the steps to create an executable census digitisation workflow, separated into the two use scenarios (User L and User C). The right-most column indicates whether semantic features were used for a specific step.

It can be concluded that basic steps of the workflow composition process can be automated. In combination with workflow templates, complex constructs can be created, likely even by users that are not experts in either document image analysis or workflow systems. Modular design (smaller workflows that can be combined) and composition aided by semantic features help to overcome obstacles. Specifically, using semantics has the following effects for the given use case:

- Improved workflow and activity discovery: Data types can have a very diverse meaning and are not sufficient for automation of workflow composition tasks. But also, a manual search profits from using an agreed ontology. A keyword-based search can lead to incomplete results since research groups, businesses, and other parties often use their own terminology that differs from others. Wrong results may be returned if keywords are ambiguous. The term “performance evaluation”,

for instance, has very different meanings in document image analysis and arts & media.

- Enablement of automated and assistive functions: The label-based semantic matching approach enables features that would fail otherwise. A matching purely based on data types of activities would lead to ambiguities that render automated functions ineffective and assistive function inefficient.
- Reusability: Enriching a workflow (or a workflow module) with meaningful semantic labels helps to make it more reusable because it can be discovered more easily by other researchers and because the purpose of the module is clearly defined.

The ontology and the model based on multiple taxonomies was expressive enough for the described use scenario. Nevertheless, there are certain limitations:

- The creation of complex templates and the composition of large workflows requires knowledge of the workflow system. Structures such as nested loops, multi-condition if-else branches, and parallelisation cannot be constructed in a fully automated way.
- The success of the automated and assistive features strongly relies on the quality of the semantic labelling and the completeness of repositories (templates, activities, and data sources).

Not all aspects of the Census 1961 project were covered in this section, but the required steps are the same: template creation, workflow discovery, assisted composition, data linkage, and validation.

Table 14 - Overview of Census digitisation workflow creation

Step	Used Features (Scenario L)	Used Features (Scenario C)	Usage of semantic features
<b>Workflow template creation</b>		Composition	Labelling
		Activity Search	Label-based
		Data Tables	Labelling
		Validation	Label matching
<b>Workflow Discovery</b>	Search by input / output data features	Search by activity features	Label-based
		Metadata	None
<b>Concretisation</b>	Automated and/or assisted concretisation	Replacing abstract activities	Label matching, Label filtering
		Text-based search	None
<b>Linking external data</b>	Automatic data conversion	Data creation	Label-based search, Label matching
<b>Filling in modules / selected activities</b>	Replacing with child activity of another workflow	Reusing a workflow module to replace an activity	None
<b>Handling data collections</b>	Automated loop creation	Connecting loop data ports	None
<b>Finalisation</b>		Validation and correction	Partial use

## 7.11 Summary

In this chapter, a project from the domain of document image analysis was used to test the expressivity of the proposed ontology and the support the workflow system implementation provides users – using a real-world setting. Hypothetical users with assumed different background knowledge were used to explore the use of the prototype system from different angles.

The findings confirm the results from Chapter 6 in that the ontology is expressive enough for modelling processing pipelines and activities related to document image analysis. The assistive features are useful and should help users in composing relevant workflows



(although future studies with real users would be beneficial to explore and remove points of friction).

This chapter strengthened the claim that objective 4. of Section 1.4 (complete framework for workflow composition and management) is indeed fulfilled. The case study covered all aspects a user would interact with when using such a framework. All essential functionality is available, with the limitations of it being a prototype. Objective 5. (real-world use cases) is also seen as completed.

The next and final chapter provides a discussion of the workflow system in context of the goals of the PhD research.

## 8 Discussion and Conclusions

At the beginning of this Thesis, workflow systems were introduced as a means to model and execute scientific experiments and processing pipelines. While scientific workflows have several advantages over traditional techniques (e.g. scripts), the modelling tools have a steep learning curve.

The use of semantic information was proposed as a means to enhance workflow systems. A taxonomy-based semantic labelling scheme with an ontology for the domain of document image analysis and recognition was presented. A prototype workflow composition and management system was developed, incorporating semantics in all components, to be able to test the respective algorithms and other aspects. Lastly, experiments and a real-world case study were discussed, assessing the semantic features for assisting users and automating tasks.

In this chapter, the presented semantic model and workflow system features are examined carefully regarding the objectives of this PhD research. The key contributions are discussed and relations to other systems are established. In addition, current limitations and future work are outlined. At the end of the chapter overall conclusions are presented.

### 8.1 Fulfilment of Objectives

The main aims of the PhD research were to investigate approaches for automation within scientific workflow systems and to develop strategies and solutions to assist with the creation and management of workflows.

The hypothesis<sup>5</sup> is proved. The proposed semantic modelling approach is powerful enough to enable useful assistive features for several tasks in workflow design/composition, use, and management (if a well-designed ontology for the target domain is used). Fully automated features are limited to small/local algorithmic interventions, but features with user interaction work well throughout.

---

<sup>5</sup> “By creating a formal model for document image analysis, by semantically annotating components and data, and by using this information algorithmically within a workflow system it is possible to automate common tasks (within such system) and assist users in workflow creation and management.” See Section 1.4.

The respective research objectives (Section 1.4) are discussed below.

### **1. To determine a suitable approach for semantic modelling**

A semantic modelling approach based on taxonomies was proposed, taking into account several aspects of workflow environments: data (objects), workflow components, workflows, and user groups. It was shown that, in combination with workflow templates, the semantic model (the ontology) is expressive enough to enable various assistive and automated features for composition and retrieval of workflows.

Technologies from the Semantic Web were chosen as modelling paradigm because they fit well with the distributed nature of workflows and because mature standards and tools exist.

By concentrating on one aspect of semantic relations (class relationships in the form of label type hierarchies), both the definition and the use of the ontology have low complexity, while meeting the demands of expressiveness.

### **2. To create a semantic model for document image analysis that covers all relevant aspects**

An ontology for document image analysis and recognition was developed, following a well-defined methodology and reflecting real-world data. Sources for the knowledge acquisition phase included existing taxonomies, text books, research project outcomes, and journal papers. The ontology was then validated (and extended) by creating an activity repository for over 100 software tools (for document image analysis) and by creating workflows for digitisation projects (e.g. Census 1961 Feasibility Study).

### **3. To develop algorithms that use semantic information to support the user in workflow-related tasks**

Algorithms were developed for semantic matching of workflow components and (semi-) automated workflow concretisation (filling in semantically fitting activities). Both allow for a smoother workflow design/composition process.

#### **4. To develop a framework that incorporates the semantic model as well as features for (semi-)automated workflow composition**

A workflow system prototype with semantic capabilities was designed and implemented. Three main modules cover all relevant aspects for semantics-enabled workflow management:

- Ontology Editor for creating and maintaining ontologies.
- Workflow Editor for composing, annotating, and viewing workflows.
- Repository Hub for aggregating workflows (or activities or data tables) and providing search functionality.

Automation and assistance were added in form of:

- Label-based search and retrieval of workflows.
- An algorithm for semantic matching of activities.
- An approach for workflow template concretisation (substituting abstract activities with concrete activities that represent software tools).
- Workflow validation (to help to complete workflows and/or enhance the quality of workflows).

#### **5. To prove the validity of the approach by applying it to real-world use cases**

The findings from the experiments (chapters 6 and 7) show that the presented model is expressive enough to describe real-world scenarios, encompassing the following elements: software tools (methods), data, data flow, control flow, semantic knowledge, and levels of abstraction. Basic workflows can be composed using semi-automated (assistive) and fully-automated methods. When used together with workflow templates (created by experts), complex structures can also be created by non-experts in the target domain and/or the workflow system.

Several workflows for a large digitisation project (1961 Census for England and Wales) were worked out and analysed. The flexibility of the proposed approach was shown by considering different types of users, approaching workflows from different angles.

The primary research objectives were achieved and, in addition, the following secondary outcomes were presented.

1. A tree-based approach was chosen to visualise the hierarchy of activities, complemented by freely arrangeable diagrams for control flow in directed acyclic graphs. This represents a low-level view, most likely to be used by workflow system experts. A higher-level view, based on UML Activity Diagram notation, was implemented in addition.
2. A mechanism to provide specialised definitions for different user groups (with different levels of domain and/or technical knowledge) can support the cooperation between experts of different domains. This encompasses additions to the ontology and to the workflow system.

## 8.2 Key Contributions

The main contributions can be summarised as:

- (1) A new approach for incorporating semantic information into workflow systems.
- (2) A new semantic model for document image analysis and recognition.
- (3) A workflow system prototype with support for semantic data and related features.
- (4) Algorithms for (semi-)automation of workflow composition and discovery.
- (5) Solutions to real-world use cases using the semantics-enriched workflow system.

The five points are discussed below.

### **Label-based semantic modelling (1)**

A new approach for incorporating a semantic layer to workflow models was proposed. Labels that are part of an ontology - based on multiple taxonomies - are used to specify the meaning of workflow components and data objects. The semantic information is treated less strictly than data types to be more forgiving regarding assignment and use of labels. Instead of a simple yes-or-no reasoning, a heuristic algorithm is used to calculate a match score when searching for workflows, components, or data.

## **Ontology (2)**

Using the proposed model, an ontology for document image analysis and recognition was engineered, considering multiple sources of information in the data collection, conceptualisation, and evaluation stages (text books, existing taxonomies, journal papers, project outcomes, software tools, and digitisation pipelines). The ontology thereby covers all important aspects of the domain:

- Research- and industry-related terms / concepts.
- Separate taxonomies for *data* and *activity*.
- *Scientific* (e.g. activity domain) and *practical* information (e.g. software licence).

## **Workflow system (3)**

A functionality-rich workflow system prototype was implemented to explore and demonstrate ways to use the semantic information (for creating and using workflows efficiently). It can be used as an example of how to add semantic features to existing workflow systems. A unique characteristic is the holistic approach of the semantic extensions, including ontology editing, labelling, composition, retrieval, and user interface.

## **Algorithms (4)**

New algorithms for semantic matching and workflow concretisation were developed. The semantic information is therein treated in a non-traditional way (less strict, resulting in a match percentage) to achieve flexibility.

## **Worked out use cases (5)**

In addition to an evaluation of each component of the system prototype (Chapter 6), worked out solutions for a large-scale project were presented in Chapter 7. The system was successfully used, in a hypothetical scenario, to create workflows for an existing digitisation project (Census 1961 for England and Wales). To test the flexibility of the model, different points of view were explored (different types of hypothetical users). All discussed aspects of the proposed approach were thereby employed: semantic annotation, (semi-)automated

workflow composition, repositories, workflow search, activity matching, and workflow validation.

The next section discusses several other approaches of incorporating and using semantic information in workflow systems.

### 8.3 Comparison to other Semantics-based Approaches

There are several workflow systems with intrinsic semantic features or semantic extensions (see Section 2.5). The most straightforward comparison that can be made is to the **ASKALON** system (see Section 2.5.3) since its workflow model was used as the basis for the presented prototype. **ASKALON** uses a very rigid semantic approach where each workflow activity can be assigned exactly one function (semantic meaning). In praxis this means, users need to know exactly what they are looking for. They then either find a matching activity or not (there is no way to discover something in-between, i.e. something that is closely related to the search terms but does not fit precisely).

Ontologies need to be created using external tools and are very specific, resembling an abstract workflow, precisely reflecting a dedicated scientific setup/experiment (judging from the examples the authors present). Reuse for different setups is therefore questionable.

Annotation of data objects is not supported (as opposed to the proposed system). Automation regarding dataflow is therefore limited to data type checks.

The proposed label-based approach is less rigid in comparison which enables more reuse and better discoverability. An extension based on weighting could emulate **ASKALON**'s functionality (see also the next section). The proposed system would then have the option of strict matches (where required) and still retain the option of flexible matches. Strictness can be useful in situations where a non-match would definitely lead to an invalid workflow.

**WINGS** (see Section 2.5.1) uses a strict constraint-based approach where workflow components are filtered out from search results if a constraint applies. The constraints are entered directly using syntax from the Semantic Web, therefore requiring expert knowledge.

This allows for more flexible semantic descriptions, but it is also more complex in terms of definition and processing. In contrast, the model proposed in this PhD research is lightweight and straightforward to use (the user interface aids the semantic annotation). Nevertheless, the two approaches are not mutually exclusive and could be combined (e.g. using labels for most cases and adding strict constraints where needed), in cases where certain combinations of activities or data should be completely forbidden. Currently, the proposed system cannot forbid any combinations. It only shows a low match score, for instance, and leaves the final decision to the user.

Gubala and Bubak's **Petri net-based system** (see Section 2.5.2) is purely automated and does not allow for user interaction. The use cases in this Thesis showed that assistive features with user actions can be an effective way of resolving conflicts where not enough semantic data is available. Real-world use will inevitably entail incomplete workflow component descriptions and other obstacles. An interactive system with some automation seems more suited to actual real-world problem scenarios. Furthermore, the Petri net system has never been developed beyond a prototype stage.

The conceptual **Semantic Grid** and the associated Dartflow (see 2.5.4) only represent a theoretical framework which seems incomplete. Iterations (or loops), for example, are not mentioned or evaluated. It is web-service-oriented and therefore not fully comparable to the proposed system with workflow templates. However, the authors describe a useful architecture for managing multiple ontologies. This could be adopted if the proposed system is to be used for multiple domains at once.

Most of the state-of-the-art systems use Semantic Web technologies. It is therefore conceivable that some semantic information could be shared across systems. Additionally, semantic features could be combined to achieve a more expressive system (although this also increases the complexity).



## 8.4 Limitations

In this section, several limitations and possible solutions are discussed.

### **Related to semantic model**

The proposed semantic model uses hierarchical relations of label types for annotating workflow components and data (low complexity, expressive enough for the discussed tasks). Complex relationships (useful when very detailed modelling is required), possibly with numerical values, cannot be reflected. However, the use of workflow templates can compensate for this and allows the pre-composition of reusable sub-workflows (that can be created by experts).

Numerical and other constraints could be added in future by extending the model using the respective features from the semantic modelling paradigms and languages (OWL, for instance). In addition, certain numeric aspects could be modelled using ranges of values (age groups for printed documents, for example: 1900 – 1950, 1950 – 2000 etc.).

### **Related to matching and search algorithms**

The flexibility of the matching algorithm (match score 0% to 100%) can lead to unsuitable (in terms of achieving the most useful results) workflows if the user fully relies solely on automated features. Where a strict approach (a yes/no semantic inference method) might fail to produce a result, the proposed approach might return an executable but ineffective workflow. A visualisation method highlighting weak semantic matches can be developed to help users to focus on potential problems (e.g. connected data ports that have differing semantic purposes).

The semantic workflow search component of the system already qualifies search results (with scores), aiding users to decide for or against the discovered workflows.

### **Related to semantic annotation**

The automated and assistive features are only as good as the data they have access to (a general limitation of any system requiring information). If activity, template, and data repositories lack semantic information, the system might not be able to produce valid

workflows. The inclusion of user interaction can help to some extent, but non-experts might not be able to complete a partial workflow either. Future workflow frameworks will require some form of quality assurance, including the proposed workflow validation method in one form or another. In addition, repositories could be moderated to guarantee annotations are as complete as possible (which is a reasonable requirement, as shown by the use case presented in this Thesis).

Adding semantic information to software methods takes time – a valuable resource. It might be difficult to convince developers to invest additional time on top of designing and implementing a method. However, having a good ecosystem of semantically labelled workflows could *save* time overall. Labelling could be a good practise similar to adding comments to source code (which also takes time in the short run and saves time in the long run).

Too much semantic annotation can also be problematic for the proposed approach, possibly leading to a dilution of information. A solution could be to add weights to labels, either in the ontology or at the time of use (e.g. semantic search). The matching algorithm can be extended to treat labels with higher weights as more important.

### **Related to user interface**

The presented prototype is not yet fully suitable for inexperienced users. Workflows can be composed using a tree visualisation of activities, but a graphical composition approach (e.g. with drag and drop functionality) needs to be added for more complete user experience (better understandability, faster composition etc.). This has, however, little relevance with respect to the objectives of this PhD research (evaluation of semantic approaches). Nevertheless, a small step has been taken in that direction by adding a UML-like visualisation component (see Section 5.4.11).

The inclusion of user groups in the semantic model and its application to the embodiment of the user interface can help to lessen learning curves and make the system more widely accessible (as already implemented for aspects such as component descriptions).

The next section discusses future applications and extensions with respect to the ontology, workflow model, and the overall system.

## 8.5 Applications and Future Work

The implementation of a complete and mature system was out of scope for this PhD research. In this section, several points are discussed which reflect the natural next steps of the different aspects of the workflow framework.

### 8.5.1 Ontology

A multi-faceted ontology for document analysis was proposed in Chapter 3. Nevertheless, extensions and repurposing are to be expected in the future in order to adapt to new developments in the domain.

#### 8.5.1.1 *Ontology Extensions*

It was pointed out that, in the context of scientific workflows, an ontology is never complete (additions and refinements for new use cases). To this end, special care was taken to make the example ontology futureproof. For instance, some high-level terms contain only one child label type at the moment but can be extended as needed. Furthermore, certain parts were omitted within this work for practical reasons. Natural languages are one example. As straightforward future extension, an ISO standard for languages (ISO 639-, -2, or -3) is recommended.

Since extensions are part of the ontology engineering process, they should also follow the complete lifecycle of the used methodology (knowledge acquisition, conceptualisation, formalisation, integration, and evaluation; see Section 3.2).

#### 8.5.1.2 *Application to other Domains*

All examples throughout this work are from within the field of document image analysis. Nonetheless, the underlying model (with concepts of activity and data) is generic and can be applied to many other areas. Hierarchically organised information, as used within an

ontology, is intrinsic to most domains in science, business, and others. Specific use cases (of workflow systems) from the literature are: meteorology, bioinformatics, and data mining.

Many concepts of the proposed ontology can be reused without modification. Examples are activity-related terms such as:

- Automation (manual, semi-automated, fully-automated)
- Platform (platform-independent, Linux, Windows etc.)
- Processing level (low-level, high-level)
- Maturity
- Data creation / transformation (acquisition, conversion, enhancement etc.)

Reusable data-related aspects include:

- Precision (ground truth, estimated, measured, random)
- Content type (data, metadata, ...)

To create an ontology in another domain, the already implemented editor can be used (for the formalisation stage of ontology engineering). Although the system currently supports only one model at a time, future iterations could treat ontologies just as an additional resource. A natural development would be to use *ontology repositories* (with support for versioning, change logs, search for suitable (base) ontologies, extension of ontologies etc).

While there are no limits to the dimensions of a semantic model, all-encompassing ontologies should be avoided for practical reasons. A mixture of different domains can lead to misunderstandings, for instance through use of similar terms with different meanings. The best praxis is a well-defined, focussed application domain and ontology.

#### 8.5.1.3 Infrastructure

In the system prototype, an ontology is stored as a local XML file. For interoperability, an ontology should be made available online and managed centrally. Similarly to what is already the case with data format standards, the management would fall to a task group, volunteers, or other consortium. An open approach is preferable, enabling users to submit suggestions and feedback (like open source software projects on GitHub). An ontology would then be

accessible via a web service (for semantic queries) or as download (in the proposed XML format). Software components should be (and are already) made available as open source and should be well-documented.

Maintainers of existing workflow systems, repository platforms, and online datasets could then be encouraged to adopt the semantic layer.

These points also correspond with the keystones of the Semantic Web (which the proposed model and system are based on), as outlined in Section 3.1:

- Building models (abstracting the world).
- Computing with knowledge (reasoning machines).
- Exchanging information (distribute, interlink, and reconcile knowledge).

An evolution of the semantic models is anticipated and useful. Possible changes to the workflow model are more fundamental and should be considered carefully. These are discussed next.

## 8.5.2 Workflow Model

The workflow model defines which basic components are available to create workflows. Possible changes include new activity types, data table extensions, and activity references.

### 8.5.2.1 Additional Activity Types

Only a limited number of types of workflow activities was introduced in this work. Additional control flow activities will be required (or at least will be useful) in order to design more complex workflows.

“*While loop*” activities can complement the already described “for loop”. The child activity can thereby be reiterated while a certain condition is met. The counterparts from programming languages – “do while” and “repeat until” – can be included in this concept.

“*Sequence*” activities are special cases of a directed acyclic graph. Child activities are processed one after the other in a predefined order. This type of activity does not have to be

part of the basic workflow model. Instead, it could be implemented in the user interface layer, based on “DAG” activities and enforcing simple sequences of vertices.

#### 8.5.2.2 Writable Data Tables

Data tables were introduced in Section 5.4.4 as read-only sources for static and/or dynamic data that can be accessed from all activities in a workflow. Such tables have columns that act like output ports that have a data collection. By extending this concept in a way that table columns can also represent input ports, data tables can be made *writable*.

Data tables with write access can be used to store workflow results persistently, which, at the moment, can only be done within an activity (invisible within the workflow). Implementing the data management within the workflow system can lead to more clarity because the whole life cycle of the data can be modelled. Figure 111 shows the template creation workflow from section 7.3 (Figure 102) extended by a data table that acts as image source and output target.

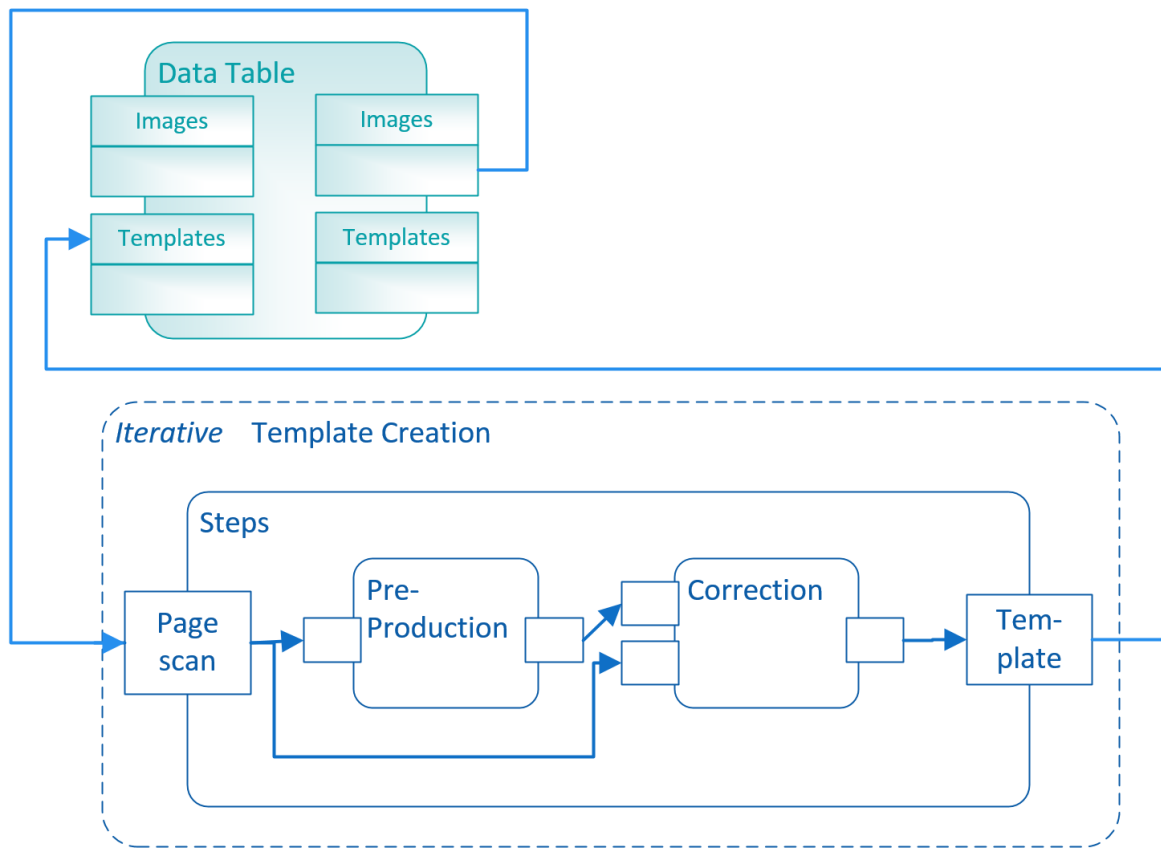


Figure 111 - Data table that acts as source and target of activity data

Another application is using a data table to store the internal state of a workflow. Usually, data flows from activity to activity. Intermediate results cannot be stored for later access within the workflow. One example is finding the maximum from a list of numerical values. In Section 7.3 this was circumvented by adding a placeholder activity “Find Maximum” which returned the list index of the maximum. Figure 112 shows a workflow that explicitly finds the maximum of a data collection of values. It uses a data table to store the internal state of the process (current maximum and value index).

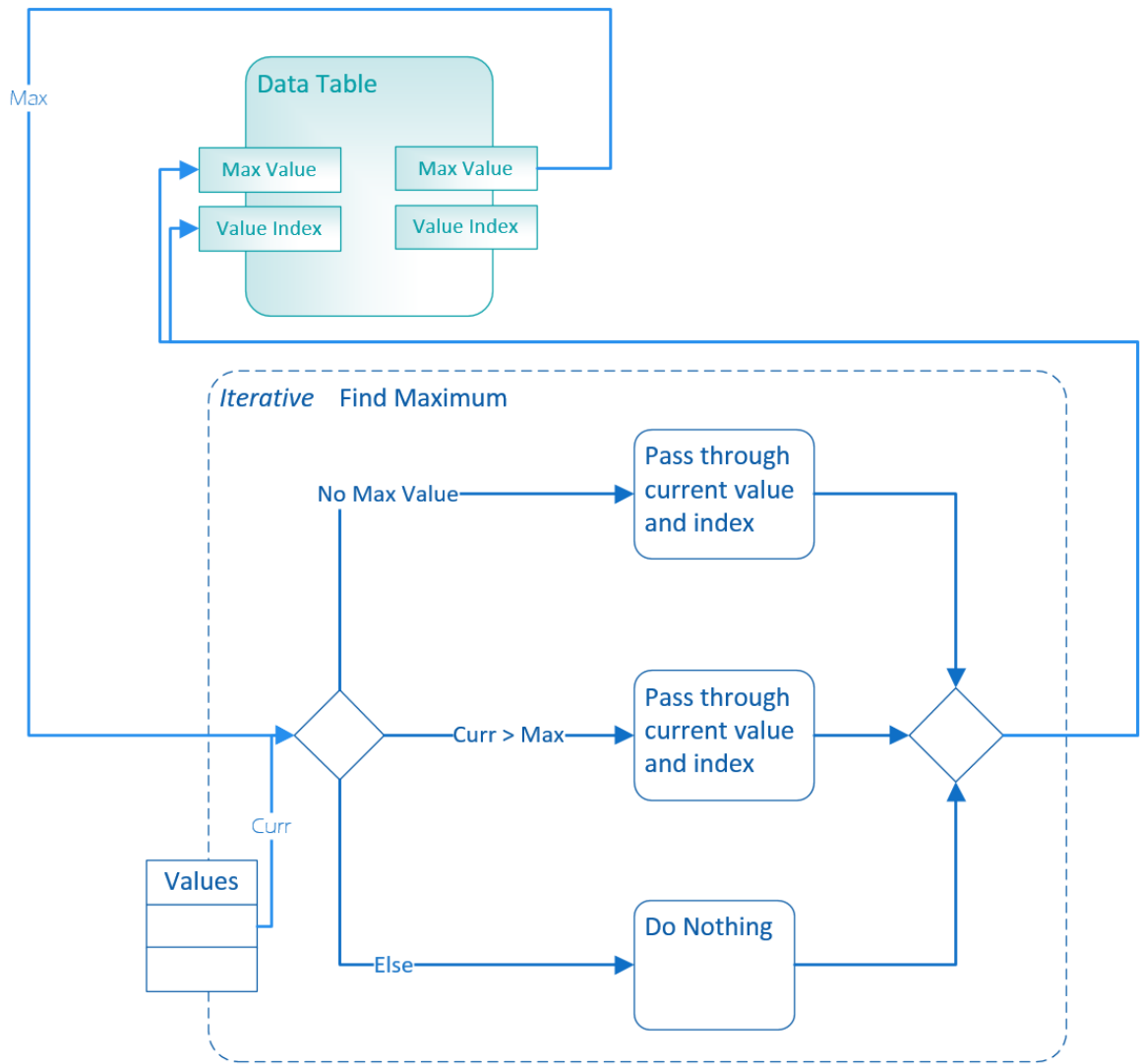


Figure 112 - Data table used as internal state for finding the maximum of values

### 8.5.2.3 Activity References

It may be necessary to be able to add one and the same abstract activity in several places within a workflow template. An example is a performance comparison workflow where the quality of two different methods (e.g. different OCR systems) is measured. It is crucial that the same performance evaluation activity is used for both methods. Figure 113 outlines the workflow. It should be noted that the evaluation must be done using two separate instances, because the quality measurement method takes results of one method and ground truth as



input and outputs a success rate (or similar). Since two methods are to be compared, the evaluation must be executed twice.

The proposed workflow model does not currently contain the concept of actual *identical* activities. A template can be designed using two *similar* abstract placeholder activities, but there is no concept of enforcing them to be replaced by the same concrete activity later.

A solution is the introduction of *activity references*. Like references in programming languages and symbolic links in file systems, they act as links to a physical object. For the example, only one abstract evaluation activity would be specified. In place of the second activity, a reference to the other would be added (see second workflow in Figure 113).

This concept is similar to sub-workflows in ASKALON [1], with the difference that these are defined independently of the main workflow. It is therefore possible to add a sub-workflow but never use it. This is not possible with activity references (the referenced activity is always part of the workflow).

Activity references have not yet been implemented as part of the prototype system. Nevertheless, since activities can be addressed via their ID, adding this feature is straightforward.

Modifications of the workflow model mostly affect the expressivity of the scientific workflows. The next subsection discusses changes to the workflow system. These are of more practical nature (usability, visualisation etc.).

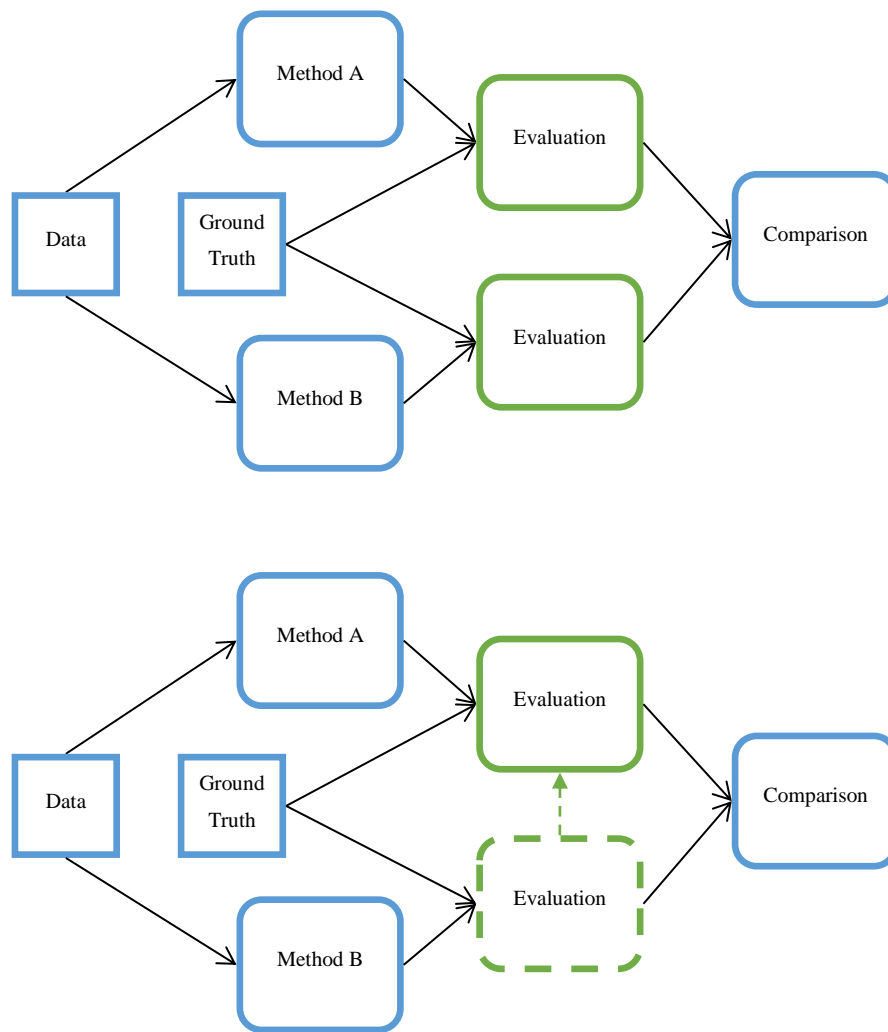


Figure 113 - Performance comparison workflow without (top) and with (bottom) activity references

### 8.5.3 Workflow System

Some additional features are not reflected in the prototype system but are outlined below.

#### 8.5.3.1 Automation

Workflow template concretisation is already implemented but depends heavily on the availability of fitting workflow components. Planning algorithms were discussed in Section 2.6 as a solution for deeper automation of workflow composition. While automating the

creation of complex workflows is unrealistic, a more powerful solution than the basic concretisation approach (Section 5.4.8) is achievable.

Given a large enough training set of existing workflows and templates, a solution based on machine learning could also be attempted. The presented matching algorithm (Section 5.4.6) can be used as part of an evaluation function.

#### *8.5.3.2 User Perspectives and User Interface*

The user interface of the presented prototype is pragmatic and was not specifically created with user experience in mind. The intention was to be able to study the underlying mechanisms which can lead to a user friendly and accessible workflow system in the future.

Certain aspects were already considered:

- Description texts and user interface elements that are customised to user groups.
- Interactive workflow composition where the system presents the user with suitable components and information on why they are suitable.
- Interactive workflow validation where the system proposed solutions and shows context sensitive user interface controls.
- Hierarchical search and matching filters where only the relevant label filter elements are shown.

For a comprehensive system, more easy-to-understand graphical interface modules need to be added. If the decision is made to extend the prototype to a complete system, existing methods should be used as role models. An UML-based method seems suitable, but refinements might be necessary. Especially for large workflows, UML activity diagrams do become convoluted and difficult to read.

User perspectives should permeate the whole system and only information that is relevant to the current user should be shown and in a way the user wants to see it. And yet, all information should still be available on demand if the circumstances require it to make informed decisions (for instance, if the simplified descriptions of activities are not sufficient to decide which activity to use, the user can opt to access the more specialised descriptions).

### 8.5.3.3 *Workflow Profiling and Optimisation*

A major advantage of workflow systems, in comparison to scripting, for instance, is that control flow and data flow are formally described. This enables advanced features such as performance prediction and workflow optimisation. To achieve this, additional metadata must be made available. This can include information on average execution time of an activity on a reference system, memory requirements, and more. Bottlenecks and weaknesses of a workflow can then be identified and possibly removed automatically.

Workflow profiling is the measurement of performance data at execution time. It does not require any of the aforementioned metadata, on the contrary, profiling can be used compile such metadata.

### 8.5.3.4 *Data Provenance*

For reproducible experiments, all data transformations and interpretations of a process must be documented. Workflow systems can be easily extended to collect this type of metadata because the dataflow between activities is well described. Semantic labels can be used to enrich the collected data. Especially attributes of the processed data may only be known at execution time.

A similar mechanism within a workflow system is logging of debug and error information. Workflow system engineers need to be able to pinpoint the origin of execution failures. Each activity should produce detailed reports in case of a fault. The execution system is responsible to collect and store this data.

### 8.5.3.5 *Distributed Resources*

All components of a workflow system can be (and arguably should be) of a distributed nature. This includes:

- Semantic model (ontology)
- Workflow/template repositories
- Activity repositories
- Data repositories
- Data

- Methods for concrete activities
- User interface

One advantage of a distributed approach is that single components can be modified/updated individually, from dedicated experts. The ontology for document image analysis, for instance, can be extended by a scientist from the domain. Other users need not to be involved.

Repositories can be implemented much more efficiently using database systems. This can be achieved significantly easier in a distributed environment. Using web services for executable activities decouples the workflow from a specific execution platform. Each software tool that is linked to an activity can be installed on a remote platform (e.g. Linux or Windows server) and can be exposed solely via web service standards. Consequently, data should be handled in a distributed way as well.

Finally, the user interface itself can be provided as a service, through a web application. While this is not a requirement (all distributed resources can be also accessed from a local application), it is a design decision that seems natural and useful. First steps in that direction have been made already with for the Taverna system [18], for instance.

The distribution of resources is also related to other research questions such as performance optimisation, fault tolerance, data provenance, trust and others.

#### *8.5.3.6 Web Ontology Language and other Formats*

For future iterations of the workflow system it can be considered to fully replace the proprietary label ontology XML format with the OWL2 format to support interoperability and extensibility. A module to read an ontology from OWL XML has not been implemented yet since it is a straightforward software engineering task and does not provide extra value for this PhD project.

It is desirable that a given workflow can be run on various workflow execution systems. This requires the conversion of the workflow description from the proposed representation to a target language. Semantic information will be lost in this process, but the structure of the workflow – the basis for execution – will be preserved.

The straightforward export target is the ASKALON system [1] since it was a main source of inspiration for the proposed workflow model. Especially the similarity of activity types and data object concepts (single vs. collection) should simplify a translation.

#### *8.5.3.7 Integration with other Systems*

A component that can be directly reused is the semantic search algorithm for retrieval of workflows from repositories. The labelling can be fully independent from the implementation specifics of the other system (labels can be stored in a separate database, for instance). Equally, the search algorithm can be applied independently.

Semantic matching and assisted workflow composition would require a deeper integration. Nevertheless, certain aspects of workflow systems seem universal, providing a starting point for such an integration. Activities and data ports are the basis for all workflows, even though the nomenclature differs. The Taverna source code, for example, contains interfaces such as “Datalink”, “DataflowInputPort”, and “Processor”.

All systems discussed in this thesis make use of the Java programming language. This is a good basis for interoperability but, given the complexity of the frameworks, a meaningful integration would require a close collaboration with the respective development teams.

A final summary is given in the next section.

## 8.6 Concluding Remarks

A new label-based approach to add a lightweight semantic layer to workflow systems and repositories was presented. A prototype was implemented which includes (but is not limited to) features for ontology editing, workflow and template composition, component retrieval, and workflow management. Algorithms for automation and semi-automation were developed, making use of the semantic model. An ontology for document image analysis and recognition was engineered and formalised using the prototype system, following an established design methodology. This allowed for a validation of the proposed system by developing workflows for a large-scale end-to-end digitisation project.

The approach was found to be sufficiently expressive for various assistive and automated features and yet to avoid the complexities of higher-level constructs of Semantic Web technologies. The hierarchical nature of the data and unambiguous definitions of terms can guide users to discover suitable semantic labels when annotating workflows or when searching for workflows, software methods, or data sources. A flexible mode of operation and distributing tasks such as ontology design and template creation, can make the workflow systems more accessible to non-experts.

Specific steps and strategies were outlined on how to extend the system or integrate the semantic features with existing workflow frameworks. The proposed semantic model, the created ontology, and the developed algorithms represent a viable foundation for a complete infrastructure surrounding workflow systems and repositories.

## References

1. Qin, J. and T. Fahringer, *Scientific workflows : programming, optimization, and synthesis with ASKALON and AWDL*. 2012, Berlin ; New York: Springer. xxi, 222 pages.
2. Foster, I., et al. *Cloud computing and grid computing 360-degree compared*. in *Grid Computing Environments Workshop, GCE 2008*. 2008. Austin, TX: Ieee.
3. Fernández-López, M., A. Gómez-Pérez, and N. Juristo, *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*, in *AAAI-97 Spring Symposium Series*. 1997, American Association for Artificial Intelligence: Stanford University, EEUU. Available from: <http://oa.upm.es/5484/>.
4. W3C, *OWL 2 Web Ontology Language Primer (Second Edition)*. 2012, Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, eds. Available from: <http://www.w3.org/TR/owl2-primer>.
5. Cohen-Boulakia, S. and U. Leser, *Search, adapt, and reuse: the future of scientific workflows*. SIGMOD Rec., 2011. **40**(2): p. 6-16.
6. Press, O.U. *Oxford Dictionaries*. [cited 2016 25/06]; Available from: <https://www.oxforddictionaries.com/>.
7. Ferilli, S., *Automatic digital Document processing and management: Problems, algorithms and techniques*. 2011: Springer Science & Business Media.
8. Kuster, M.W. *The Four and a Half Challenges of Humanities Data*. in *2011 International Conference on Document Analysis and Recognition*. 2011. IEEE.
9. Le Bourgeois, F., et al. *Document images analysis solutions for digital libraries*. in *Document Image Analysis for Libraries, 2004. Proceedings. First International Workshop on*. 2004. IEEE.



10. Hollingsworth, D., *The Workflow Reference Model*. 1995, Workflow Management Coalition. Available from: <ftp://ftp.ufv.br/dpi/mestrado/Wkflow-BPM/The%20Workflow%20Reference%20Model.pdf>.
11. Fox, G.C. and D. Gannon, *Special Issue: Workflow in Grid Systems: Editorials*. *Concurr. Comput. : Pract. Exper.*, 2006. **18**(10): p. 1009-1019.
12. Talia, D., *Workflow Systems for Science: Concepts and Tools*. ISRN Software Engineering, 2013.
13. Sonntag, M., D. Karastoyanova, and E. Deelman, *Bridging the gap between business and scientific workflows: humans in the loop of scientific workflows*, in *e-Science (e-Science)*, 2010 IEEE Sixth International Conference on. 2010, IEEE. p. 206-213.
14. Buyya, J.Y.R., *A Taxonomy of Workflow Management Systems for Grid Computing*. 2005, University of Melbourne: Melbourne, Australia. Available from: <http://www.aiai.ed.ac.uk/~jessicac/project/IRR/5-workflow-taxonomy-in-grid-2005.pdf>.
15. Corcho, O., et al., *Common motifs in scientific workflows: An empirical analysis*, in *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*. 2012, IEEE Computer Society. p. 1-8.
16. Barker, A. and J.V. Hemert, *Scientific workflow: a survey and research directions*, in *Proceedings of the 7th international conference on Parallel processing and applied mathematics*. 2008, Springer-Verlag: Gdansk, Poland. p. 746-753.
17. Oinn, T., et al., *Taverna: a tool for the composition and enactment of bioinformatics workflows*. *Bioinformatics*, 2004. **20**(17): p. 3045-3054.
18. Oinn, T., et al., *Taverna: lessons in creating a workflow environment for the life sciences*. *Concurrency and Computation: Practice and Experience*, 2006. **18**(10): p. 1067-1100.

19. Ludaescher, B., et al., *Scientific workflow management and the Kepler system: Research Articles*. *Concurr. Comput. : Pract. Exper.*, 2006. **18**(10): p. 1039-1065.
20. Majithia, S., et al. *Triana: A graphical web service composition and execution toolkit*. in *Web Services, 2004. Proceedings. IEEE International Conference on*. 2004. IEEE.
21. Deelman, E., et al., *Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems*. *Scientific Programming*, 2005. **13**(3).
22. Deelman, E., et al., *Pegasus, a workflow management system for science automation*. *Future Gener. Comput. Syst.*, 2015. **46**(C): p. 17-35.
23. Curcin, V. and M. Ghanem. *Scientific workflow systems-can one size fit all?* in *2008 Cairo International Biomedical Engineering Conference*. 2008. IEEE.
24. Belhajjame, K., et al. *Metadata management in the taverna workflow system*. in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. 2008. IEEE.
25. Turi, D., et al. *Taverna workflows: Syntax and semantics*. in *e-Science and Grid Computing, IEEE International Conference on*. 2007. IEEE.
26. Wolstencroft, K., et al., *The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud*. *Nucleic Acids Res*, 2013. **41**(Web Server issue): p. W557-61.
27. Neudecker, C., et al. *An experimental workflow development platform for historical document digitisation and analysis*. in *Proceedings of the 2011 workshop on historical document imaging and processing*. 2011. ACM.
28. Neudecker, C. *Evaluation Framework and Taverna*. 2011. Available from: <http://impactocr.wordpress.com/2011/07/12/evaluation-framework-and-taverna-with-clemens-neudecker/>.

29. Tan, W., et al., *A comparison of using Taverna and BPEL in building scientific workflows: the case of caGrid*. *Concurrency and Computation: Practice and Experience*, 2010. **22**(9): p. 1098-1117.
30. *Ptolemy Project*. 2016 [cited 2016 26/06]; Available from: <http://ptolemy.eecs.berkeley.edu/ptolemyII/>.
31. Bowers, S. and B. Ludäscher, *An Ontology-Driven Framework for Data Transformation in Scientific Workflows*, in *Data Integration in the Life Sciences: First International Workshop, DILS 2004, Leipzig, Germany, March 25-26, 2004. Proceedings*, E. Rahm, Editor. 2004, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 1-16.
32. Taylor, I., *Triana Generations*, in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. 2006, IEEE Computer Society. p. 143.
33. TC, O.W.S.B.P.E.L.W. *Web Services Business Process Execution Language Version 2.0*. [cited 2016 26/06]; Available from: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>.
34. Deelman, E., et al., *Workflows and e-Science: An overview of workflow system features and capabilities*. *Future Generation Computer Systems*, 2009. **25**(5): p. 528-540.
35. Cardiff, U.o. *Triana GitHub repository*. 2017 [cited 2017 16/12]; Available from: <https://github.com/CSCSI/Triana>.
36. Gil, Y., et al. *Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows*. in *OWLED*. 2006.
37. Fahringer, T., J. Qin, and S. Hainzer. *Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language*. in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005*. 2005. IEEE.

38. Wiczorek, M., R. Prodan, and T. Fahringer, *Scheduling of scientific workflows in the ASKALON grid environment*. ACM SIGMOD Record, 2005. **34**(3): p. 56-62.
39. Pllana, S., J. Qin, and T. Fahringer. *Teuta: A Tool for UML based composition of scientific grid workflows*. in *1st Austrian Grid Symposium, Schloss Hagenberg, Austria*. 2005.
40. Salimifard, K. and M. Wright, *Petri net-based modelling of workflow systems: An overview*. European journal of operational research, 2001. **134**(3): p. 664-676.
41. Hoheisel, A., *User tools and languages for graph-based Grid workflows*. Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1101-1113.
42. Guan, Z., et al., *Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface*. Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1115-1140.
43. Mendling, J. and M. Nüttgens, *EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC)*. Information Systems and e-Business Management, 2006. **4**(3): p. 245-263.
44. Eswaran, S., et al., *Adapting and evaluating commercial workflow engines for e-Science*, in *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*. 2006, IEEE. p. 20-20.
45. Juhnke, E., et al. *SimpleBPEL: Simplified Modeling of BPEL Workflows for Scientific End Users*. in *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. 2010. IEEE.
46. List, B. and B. Korherr, *An evaluation of conceptual business process modelling languages*, in *Proceedings of the 2006 ACM symposium on Applied computing*. 2006, ACM. p. 1532-1539.

47. Ivanova, V. and L. Stromback. *Creating Infrastructure for Tool-Independent Querying and Exploration of Scientific Workflows*. in *E-Science (e-Science), 2011 IEEE 7th International Conference on*. 2011. IEEE.
48. Moreau, L., et al., *The open provenance model core specification (v1. 1)*. Future generation computer systems, 2011. **27**(6): p. 743-756.
49. Plankensteiner, K., et al., *Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures*. Journal of Grid Computing, 2013. **11**(3): p. 429-455.
50. Roure, D.D., C. Goble, and R. Stevens, *The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows*. Future Generation Computer Systems, 2008. **25**: p. 7.
51. De Roure, D., et al., *Towards open science: the myExperiment approach*. Concurrency and Computation: Practice and Experience, 2010. **22**(17): p. 2335-2353.
52. Mates, P., et al., *CrowdLabs: Social Analysis and Visualization for the Sciences*, in *Scientific and Statistical Database Management: 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*, J. Bayard Cushing, J. French, and S. Bowers, Editors. 2011, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 555-564.
53. *Universal Description, Discovery, and Integration - UDDI*. [cited 2016 24/09]; Available from: <http://uddi.xml.org/>.
54. Littauer, R., et al., *Trends in use of scientific workflows: insights from a public repository and recommendations for best practice*. International Journal of Digital Curation, 2012. **7**(2): p. 92-100.
55. Gil, Y., et al. *Expressive reusable workflow templates*. in *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*. 2009. IEEE.

56. Hauder, M., Y. Gil, and Y. Liu. *A framework for efficient data analytics through automatic configuration and customization of scientific workflows*. in *E-Science (e-Science), 2011 IEEE 7th International Conference on*. 2011. IEEE.
57. Gubala, T., et al. *Semantic composition of scientific workflows based on the petri nets formalism*. in *2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*. 2006. IEEE.
58. Nadarajan, G., Y.-h. Chen-Burger, and J. Malone. *Semantic-Based Workflow Composition for Video Processing in the Grid*. in *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*. 2006. IEEE.
59. Siddiqui, M., A. Villazon, and T. Fahringer. *Semantic-based on-demand synthesis of grid activities for automatic workflow generation*. in *e-Science and Grid Computing, IEEE International Conference on*. 2007. IEEE.
60. *Protégé - A free, open-source ontology editor and framework for building intelligent systems*. [cited 2015 24/03].
61. Xing, W., M.D. Dikaiakos, and R. Sakellariou. *A core grid ontology for the semantic grid*. in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*. 2006. IEEE.
62. Gil, Y., et al., *A semantic framework for automatic generation of computational workflows using distributed data and component catalogues*. *Journal of Experimental & Theoretical Artificial Intelligence*, 2011. **23**(4): p. 389-467.
63. Gil, Y., et al., *Wings: Intelligent workflow-based design of computational experiments*. *IEEE Intelligent Systems*, 2011. **26**(1): p. 62-72.
64. Gil, Y., et al. *WINGS*. [cited 2009 24/09]; Available from: <http://www.wings-workflows.org/>.

65. *Apache Jena*. [cited 2016 24/09]; Available from: <https://jena.apache.org/>.
66. Kryza, B., et al., *Grid organizational memory—provision of a high-level Grid abstraction layer supported by ontology alignment*. *Future Generation Computer Systems*, 2007. **23**(3): p. 348-358.
67. Wu, Z. and H. Chen, *Semantic grid : model, methodology and applications*. *Advanced topics in science and technology in China*. 2008, Berlin: Springer. xiii, 230 p.
68. *OWL-S*. 2004 [cited 2016 25/09]; Available from: <https://www.w3.org/Submission/OWL-S/>.
69. Russell, S.J. and P. Norvig, *Artificial intelligence : a modern approach*. Prentice Hall series in artificial intelligence. 1995, Englewood Cliffs, N.J.: Prentice Hall. xxviii, 932 p.
70. Ambite, J.L. and D. Kapoor. *Automatically Composing Data Workflows with Relational Descriptions and Shim Services*. 2007. Berlin, Heidelberg: Springer Berlin Heidelberg.
71. Anthony Barrett, D.S.W., *Partial-order planning*. *Artificial Intelligence*, 1994. **67**(1): p. 71-112.
72. Curcin, V., P. Missier, and D. De Roure, *Simulating Taverna workflows using stochastic process algebras*. *Concurrency and Computation: Practice and Experience*, 2011. **23**(16): p. 1920-1935.
73. Jansen-Vullers, M. and M. Netjes. *Business process simulation—a tool survey*. in *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark*. 2006.

74. Chen, W. and E. Deelman. *Workflowsim: A toolkit for simulating scientific workflows in distributed environments*. in *E-Science (e-Science), 2012 IEEE 8th International Conference on*. 2012. IEEE.
75. Lamiroy, B. and D. Lopresti. *An open architecture for end-to-end document analysis benchmarking*. in *2011 International Conference on Document Analysis and Recognition*. 2011. IEEE.
76. *IMPACT Centre of Competence*. 2013 [cited 2013 09/11]; Available from: <http://www.digitisation.eu/>.
77. Hitzler, P., M. Krotzsch, and S. Rudolph, *Foundations of semantic web technologies*. 2009: CRC Press.
78. Uschold, M. and M. Gruninger, *Ontologies: Principles, methods and applications*. *The knowledge engineering review*, 1996. **11**(02): p. 93-136.
79. Fernandez-Lopez, M. and A. Gomez-Perez, *Overview and analysis of methodologies for building ontologies*. *Knowl. Eng. Rev.*, 2002. **17**(2): p. 129-156.
80. Uschold, M., *Building Ontologies: Towards a Unified Methodology*, in *Expert Systems 96*. 1996: Cambridge, UK.
81. M. Gruninger, M.S.F., *Methodology for the design and evaluation of ontologies*, in *International Joint Conference on Artificial Intelligence*. 1995.
82. Swartout, B., et al. *Toward distributed use of large-scale ontologies*. in *Proc. of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems*. 1996.
83. Knight, K. and S.K. Luk, *Building a large-scale knowledge base for machine translation*, in *Twelfth national conference on Artificial intelligence*. 1994, ACM: Seattle, Washington, USA. p. 773-778.



84. Obrst, L., et al., *The evaluation of ontologies*, in *Semantic web*. 2007, Springer. p. 139-158.
85. Flahive, A., et al., *A methodology for ontology update in the semantic grid environment*. *Concurrency and Computation: Practice and Experience*, 2015. **27**(4): p. 782-808.
86. O’Gorman, L. and R. Kasturi, *Document Image Analysis*. 1997: IEEE Computer Society.
87. *Analyzed LAYOUT and Text Object - ALTO*. 2018 [cited 2018 04/12/2018]; Available from: <https://altotml.github.io/>.
88. Pletschacher, S. and A. Antonacopoulos. *The PAGE (page analysis and ground-truth elements) format framework*. in *Pattern Recognition (ICPR), 2010 20th International Conference on*. 2010. IEEE.
89. Vincent, L. *Google Book Search: Document Understanding on a Massive Scale*. in *icdar*. 2007.
90. Sankar, K.P., et al. *Digitizing a million books: Challenges for document analysis*. in *International Workshop on Document Analysis Systems*. 2006. Springer.
91. Pletschacher, S., C. Clausner, and A. Antonacopoulos, *Europeana newspapers OCR workflow evaluation*, in *2015 Workshop on Historical Document Imaging and Processing (HIP2015)*. 2015, ACM Digital Library: Nancy, France. p. 39-46. Available from: <http://usir.salford.ac.uk/37655/>.
92. *ACM Computing Classification System*. 2012 [cited 2016 01/05]; Available from: <http://www.acm.org/about/class/2012>.
93. Jähne, B., *Digital Image Processing*. 2005: Springer Berlin Heidelberg.
94. *Succeed first online report on available tools - Annex 1*. 2013, University of Alicante.

95. *IMPACT Project 2011 Publishable summary*. 2012. Available from: <http://www.impact-project.eu/documents/>.
96. *Europeana Newspapers Project*. [cited 2016 01/06]; Available from: <http://www.europeana-newspapers.eu/>.
97. Fahringer, T., et al., *Askalon: A development and grid computing environment for scientific workflows*, in *Workflows for e-Science*. 2007, Springer. p. 450-471.
98. *Selecting a development approach*. 2008. Available from: <https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf>.
99. *OWL 2 Web Ontology Language XML Schema*. [cited 2016 01/10]; Available from: <http://www.w3.org/2009/09/owl2-xml.xsd>.
100. Lab, P.R.a.I.A.R. *PRIMA Tools*. 2018 [cited 2018 10/12/2018]; Available from: <https://www.primaresearch.org/tools>.
101. Witten, I.H., E. Frank, and M.A. Hall, *Data mining : practical machine learning tools and techniques*. 3rd ed. Morgan Kaufmann series in data management systems. 2011, Burlington, MA: Morgan Kaufmann. xxxiii, 629 p.
102. *Weka 3: Data Mining Software in Java*. [cited 2016 26/07]; Available from: <http://www.cs.waikato.ac.nz/ml/weka/>.
103. *Office for National Statistics*. [cited 2016 29/10]; Available from: <https://www.ons.gov.uk/>.
104. *Jisk*. [cited 2016 29/10]; Available from: <https://www.jisc.ac.uk/>.
105. Clausner, C., et al., *Creating a Complete Workflow for Digitising Historical Census Documents: Considerations and Evaluation*, in *Proceedings of the 4th International*

- Workshop on Historical Document Imaging and Processing*. 2017, ACM: Kyoto, Japan. p. 83-88.
106. Clausner, C., et al., *Unearthing the recent past : digitising and understanding statistical information from census tables*, in *Second International Conference on Digital Access to Textual Cultural Heritage (DATECH 2017)*. 2017, ACD Digital Library: Goettingen, Germany. p. 149-154. Available from: <http://usir.salford.ac.uk/43461/>.
  107. Clausner, C., S. Pletschacher, and A. Antonacopoulos. *Aletheia-an advanced document layout and text ground-truthing system for production environments*. in *2011 International Conference on Document Analysis and Recognition*. 2011. IEEE.
  108. *ABBYY FineReader Engine 11*. [cited 2016 29/10]; Available from: <http://www.abbyy.com/ocr-sdk>.
  109. Smith, R., *An Overview of the Tesseract OCR Engine*, in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*. 2007, IEEE Computer Society. p. 629-633.
  110. Clausner, C., S. Pletschacher, and A. Antonacopoulos. *Scenario driven in-depth performance evaluation of document layout analysis methods*. in *2011 International Conference on Document Analysis and Recognition*. 2011. IEEE.

## Appendix A – Label type hierarchies of the developed ontology (document image analysis)

Activity							
Automation	Automation	Manual					
		Automated					
		Machine assisted / user-driven					
Licence	Licence	Free	Non-commercial				
			Paid for	Pay once			
		Volume					
		Subscription					
		Open Source					
		Platform	Platform	Windows			
Mac OS							
Linux							
Platform independent	Java						
	Web						
Mobile	iOS						
	Android						
Activity Domain	Activity Domain	Computing	Visual Computing	Image and video processing	Geometric image/video processing		
						Pixel-based image/video processing	
					Content analysis and recognition	Text and symbol recognition	OCR
							Math. expr. rec.
							Date recognition
						Table / form analysis and recognition	
						Chart recognition	
						Map and plan reading	
						Object / shape recognition	
					Face recognition		
					Layout analysis		
					Computer graphics		
					Text processing	Natural language processing	Language identification
							Sentiment mining
							Summarising
							Part-of-speech tagging
Named entity recognition							
Machine learning							
Information Management							
	Data retrieval						
Performance evaluation							
	Comparative performance analysis						
	In-depth performance analysis						

			Forensic studies			
	Processing Level					
		Low-level processing				
		High-level processing				
			Detection / Identification			
				Verification / authentication		
			Classification / recognition			
			Understanding			
	Data Creation / Transformation					
		Acquisition				
		Conversion				
		Segmentation / tokenisation				
		Enhancement				
		Enrichment				
			Annotation / labelling			
		Information extraction				
		Visualisation / presentation				
	Adaptability / Applicability					
		Configurable				
		Trainable				
			Supervised			
			Unsupervised			
		Interactive				
		Generic / unconstraint				
	Maturity					
		Stable				
		Experimental				
		Industrial				
<b>Data Object</b>						
	Original Source					
		Produced data				
			Physical source medium			
				Paper document		
					Book	
					Newspaper	
					Magazine	
					Journal	
				Whiteboard / blackboard		
				Poster		
			Virtual source medium			
				World Wide Web		
		Captured data				
			Real / natural scenes			
				3D scenes		
	Age					
		Historical				
			Medieval			
		Contemporary				
		Ancient				
	Physical Production Method					
		Manual				
		Machine				
			Printed			
				Typeset		
				Computer printout		
			Typewritten			
	Acquisition / Replication Method					
		Analog / physical to digital				
			Scanning			
			Camera			
		Copied				
			Photocopy			
			Carbon copy			
			Microfilm / microfiche			
			Faxed			
		Synthesis				
	Precision					
		Ground Truth / gold standard				
		Measured				

		Estimated				
		Random				
		Fuzzy				
	Content Type	Data				
		Metadata				
			Quality			
				Performance Information		
			Features			
			Structure			
				Table of contents		
			Annotations			
			Authorship			
			Spatial			
				Location		
		Settings				
		Model				
		Lexicon / index				
		Corpus / database				
	Content Encoding	Textual				
			Annotated			
		Structured				
		Raster image				
			Colour Image			
			Bitonal			
		Mathematical / geometrical				
			Vector-based			
				Stroke-based		
			Polygonal			
	Content of Interest	Visual content				
			Text			
			Graphical			
				Separators		
				Barcode / QR code		
			Image			
				Photograph		
					Person(s)	
						Face(s)
				Drawing		
			Mixed / composite content			
				Tables / forms		
				Charts		
				Maps / plans		
				Mathematical expression		
				Chemical notation		
				Musical notation		
	Data Granularity	Physical / visual granularity				
			Document-related			
				Double-page		
				Page		
				Region / Zone		
				Text line		
				Word		
				Glyph		
			Natural language-related			
				Sentence		
				Token / chunk		
				Syllable		
		Logical granularity				
			Document-related			
				Document		
				Chapter		
				Section		
				Article		
				Paragraph		
			Table			

				Column		
				Row		
				Cell		
	Data Condition					
		Noisy				
			Speckles			
					Salt-and-pepper noise	
			Clutter			
					Thresholding-related noise	
		Production-related				
			Document characteristics			
					Pasted clippings	
					Textured paper	
					Uneven character spacing	
					Narrow border	
					Low paper-to-content contrast	
					Halftoning	
					Dithering	
		Document faults				
			Bleed-through			
			Ink from facing page			
			Smearred ink			
			Touching characters			
					Horizontally	
					Vertically	
			Uneven ink distribution			
			Filled-in characters			
			Sort shoulder artefacts			
			Broken characters			
			Faint characters			
			Blurred characters			
			Non-straight text lines			
		Wear / use				
			Medium damage			
					Folds	
					Tears	
					Holes	
						Punch holes
						Unintended holes
					Missing parts	
					Stains	
					Scratches	
					Staples	
			Additions			
					Visible repairs	
						Paper repairs
						Clear tape
					Informative	
						Annotations
						Stamps
					Corrections	
						Manual corr.
		Ageing				
			Warping			
			Discolouration			
					Global	
					Edges	
			Disintegration			
					Uneven edges	
			Mould			
			Fading content			
		Acquisition / conversion-related issues				
			Geometric issues			
					Skew	
						Global
						Non-uniform
					90-degree rotation	
					Upside down	
					Perspective distortions	

					Page curl		
					Content / background-related		
					Incomplete capture		
					Tight / narrow margins		
					Included other objects		
						Part of proceeding or succeeding obj.	
						Medium structure (e.g. book cover)	
						Paper clips	
						Fingers	
						Insects	
						Background (e.g. scan bed)	
					Method flaws		
					Imaging-related		
						Show-through	
						Uneven illumination	
							Shadows
						Out-of-focus	
						Low contrast	
						Missing / changed content	
							Due to threshld.
					Data Attributes / Properties		
					Language		
					Natural language		
						English	
					Mixed languages		
					Document-related		
					Visual properties		
					Text-related		
					Script		
						Braille	
						Latin	
					Font-related		
						Cursive	
						Monospace	
						Hand / typeface class	
							Blackletter
							Antiqua
							Manuscript
						Decorated text	
							Flourishes
							Multi-colour
							Reverse video
						Multi-font	
							Mixed typeface
							Mixed sizes
					Drop caps		
					Columns		
						One column	
						Two columns	
						Multi-column	
					Rotated content		
					Complex backgrnd		
						Watermarks	
						Impressions / embossings	
					Illustrations		
						Multi-coloured	
					Decorations		
						Frames / borders	
					Line drawing / line-art		
					CAPTCHAs		
					Structural		
					Running titles		
					Footnotes		
					Bibliographic reference		
					Topic		
					Economy		
					Financial / business		
						Bank checks	
						Invoices	



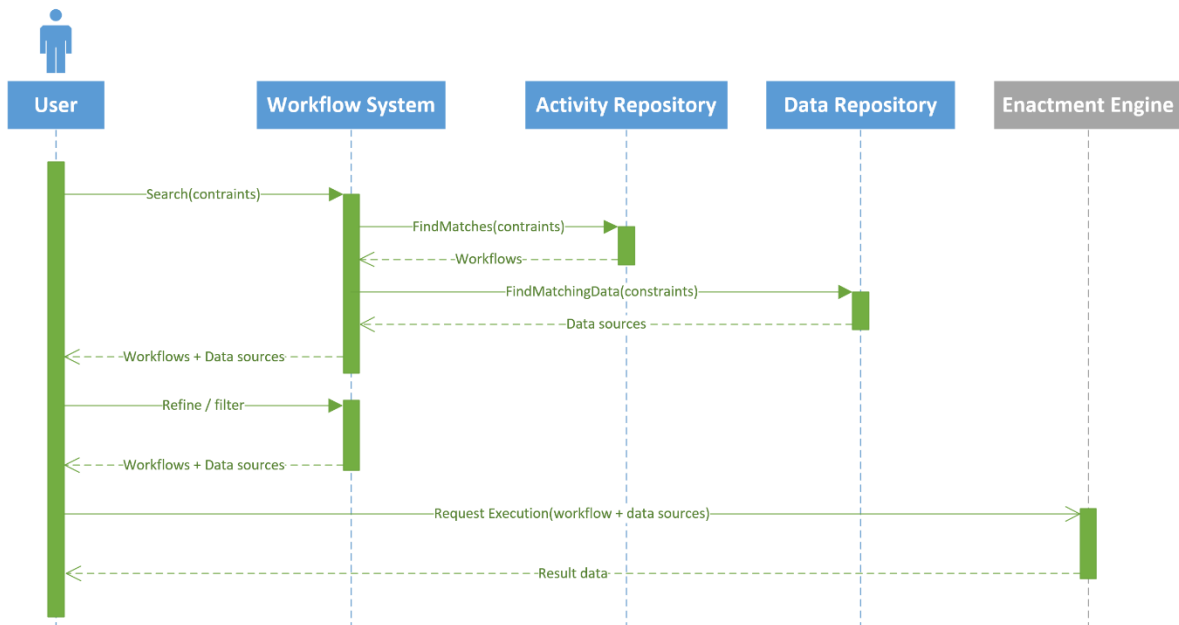
				Bank notes / paper currency	
		Social science / environmental			
			Maps		
				Topographical maps	
				Road maps	
				Land use maps	
			Traffic and automotive		
				Number plates	
				Traffic signs	
		Science and Engineering			
			Architectural		
				Floor plans	
				Architectural drawings	
			Medical		
			Engineering drawings		
			Patents		
		Media / entertainment			
			Advertisements		
		Computing			

# Appendix B – List of All Implemented Interfaces and Classes

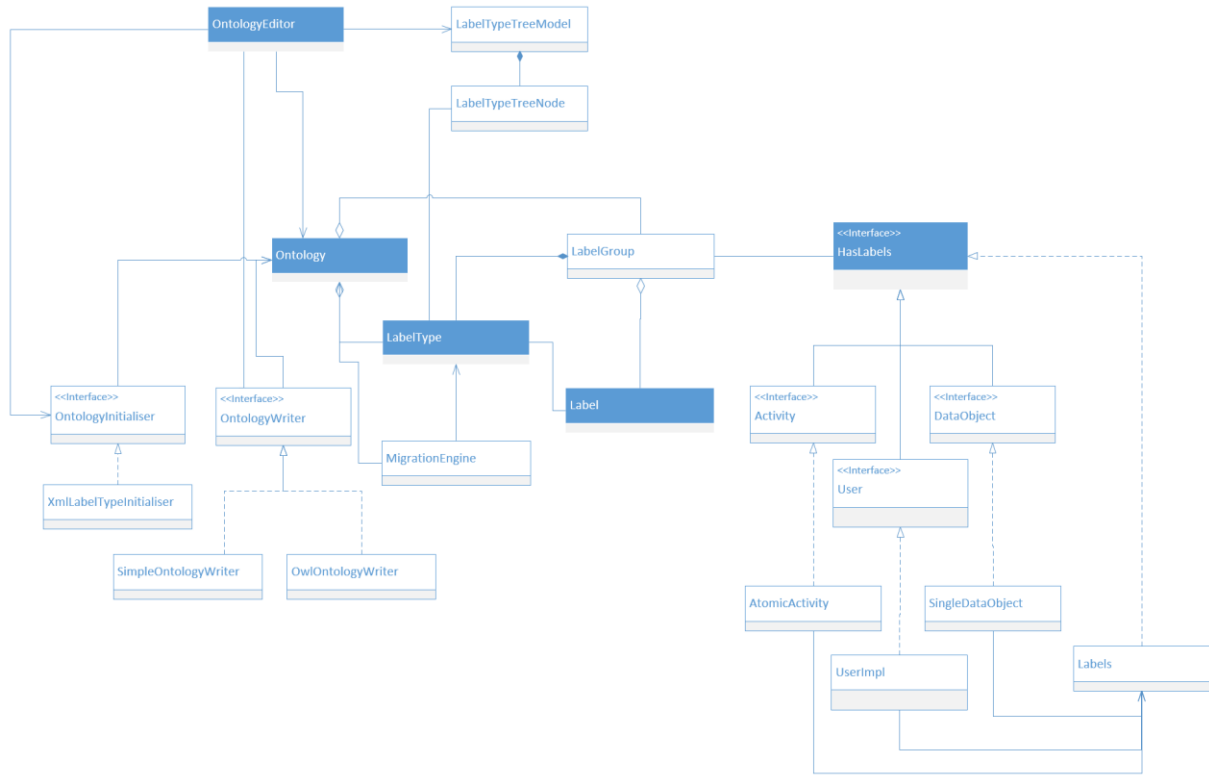
The source code is available on GitHub:

<https://github.com/PRImA-Research-Lab/semantic-labelling>

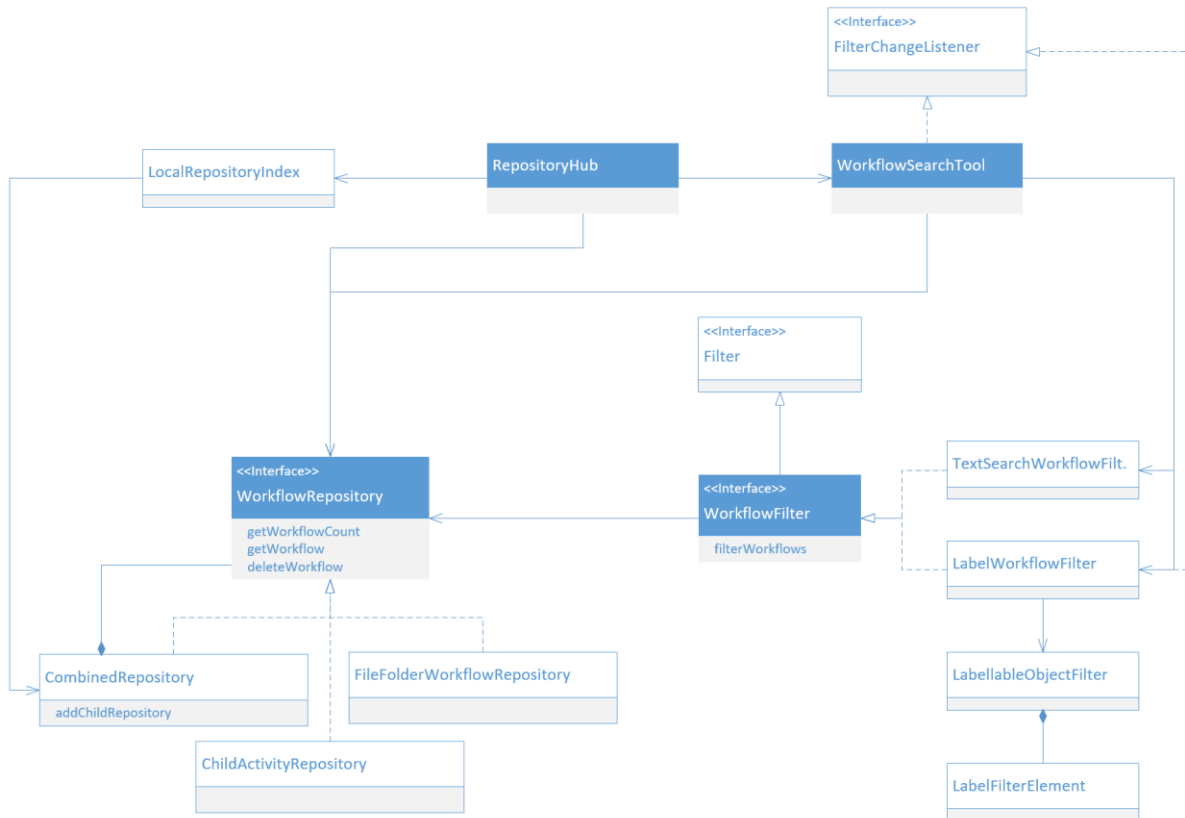
## Sequence Diagram for Workflow System Use



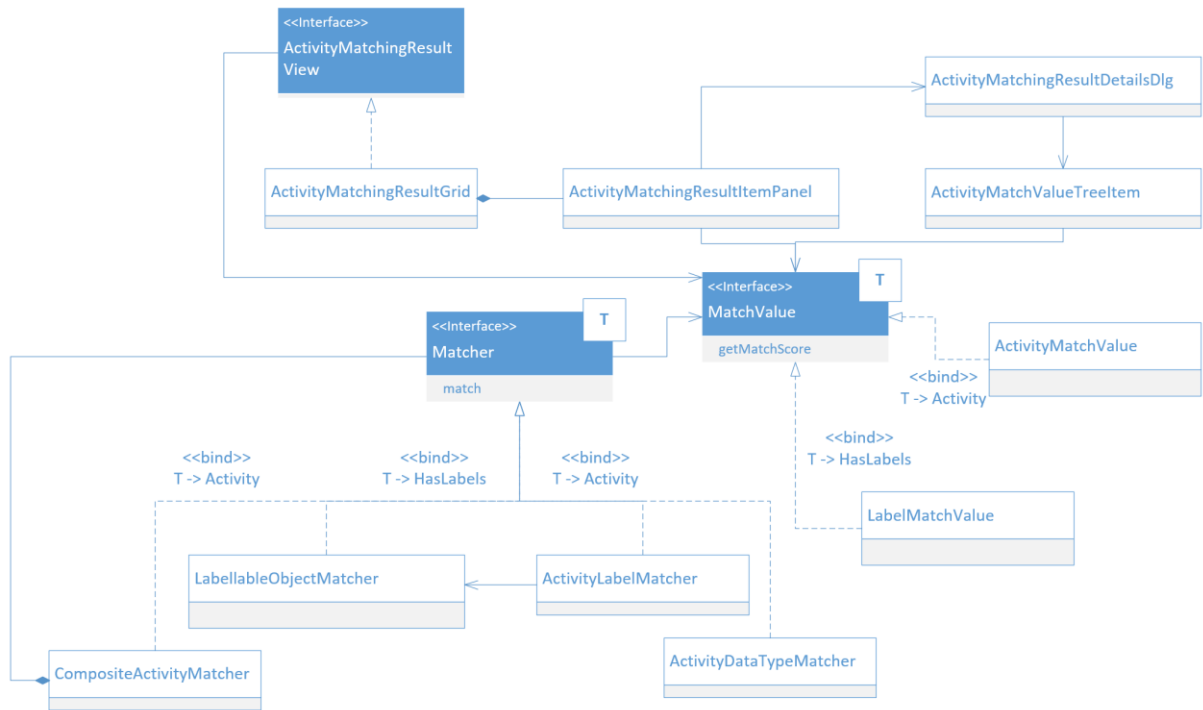
# Extended UML diagram for ontology-related classes and interfaces



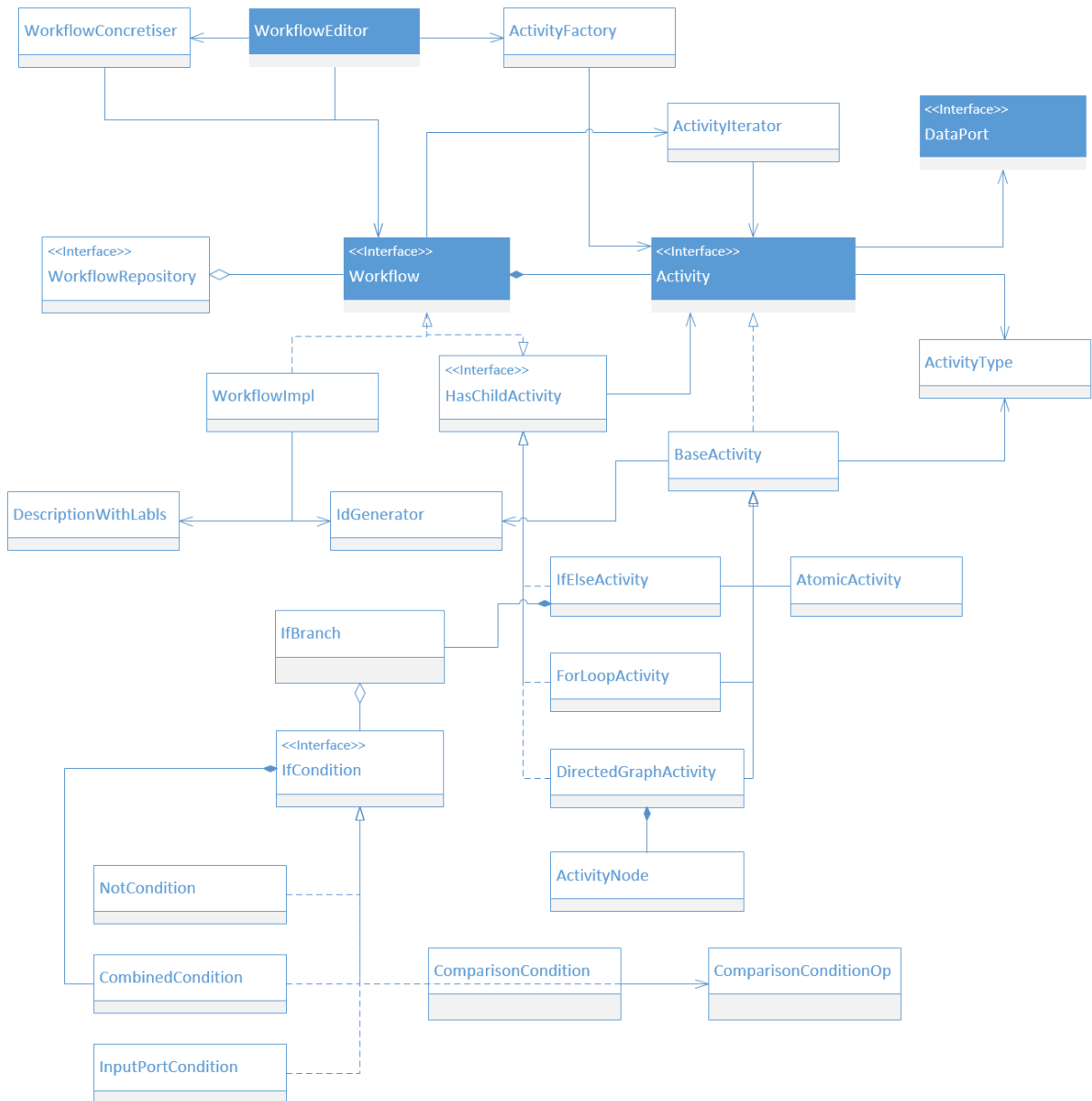
# Extended UML diagram for workflow repository-related classes and interfaces



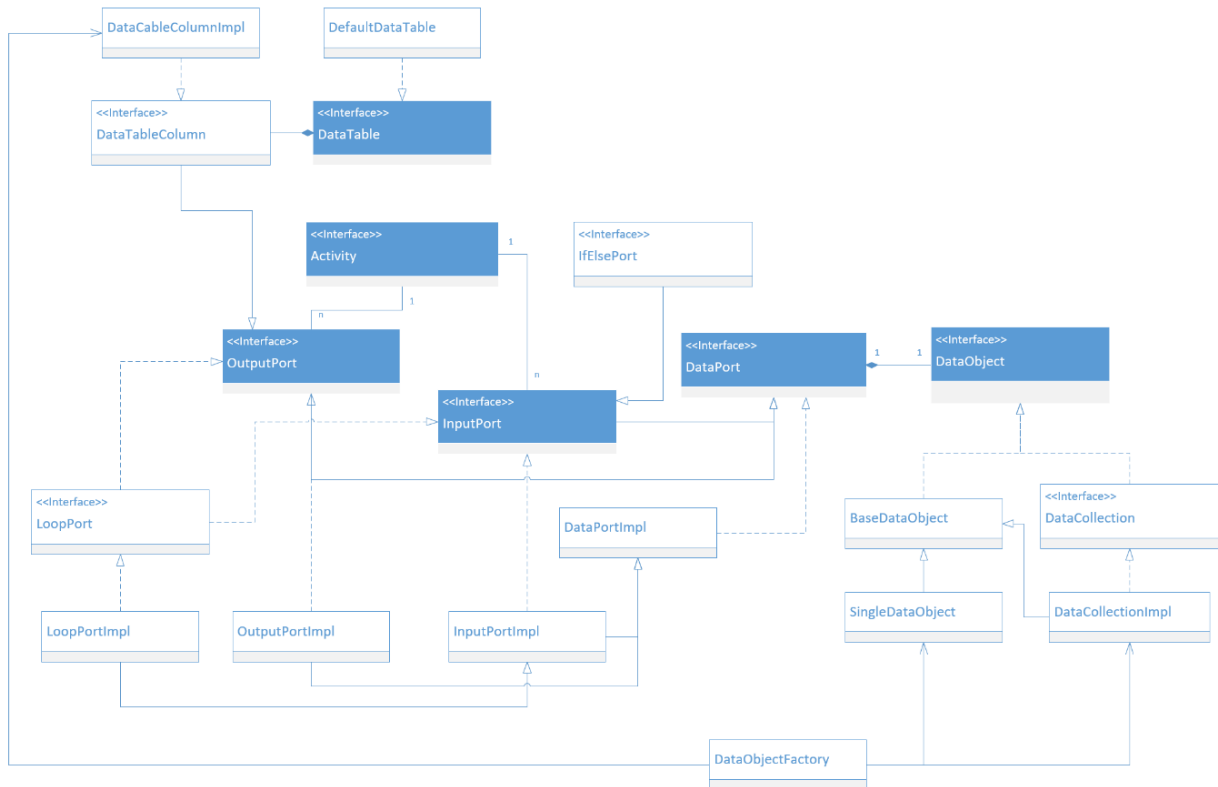
# Extended UML diagram for matching-related classes and interfaces



## Extended UML diagram for workflow-related classes and interfaces



## Extended UML diagram for data-related classes and interfaces



## Alphabetic List of Classes and Interfaces

This is an alphabetic overview of all interfaces and classes that have been implemented (important ones are highlighted):

#	Class / Interface	Short Description
1	Activity	Interface for workflow activities (tools, algorithms, control flow, ...) with input and output data ports.
2	ActivityDataTypeMatcher	Matcher implementation using the data type of activity ports.
3	ActivityDetails	Panel with input fields for basic properties of a workflow activity.
4	ActivityFactory	Factory for different types of activity objects (atomic, loop, if-else etc.).
5	ActivityIterator	Iterator for an activity tree (depth-first traversal)
6	ActivityLabelMatcher	Matcher implementation comparing semantic labels of the activity itself and all data ports.
7	ActivityMatchingResultDetailsDialog	Dialogue with tree for all match values of an activity matching.

#	Class / Interface	Short Description
8	ActivityMatchingResultGrid	Grid view with scrolling, showing all items of a matching result.
9	ActivityMatchingResultItemPanel	A clickable panel for a single matching result item.
10	ActivityMatchingResultItemPanel. ActivityMatchingResultItemListener	Listener interface for result items of workflow search or activity matching.
11	ActivityMatchingResultView	Interface for user interface views showing activity matching results.
12	ActivityMatchValue	MatchValue implementation for Activity matching results (has sub values).
13	ActivityMatchValueTreeItem	DefaultMutableTreeNode extension for result items of an activity matching.
14	ActivityTreeNode	Abstract tree node for workflow activities.
15	ActivityType	Type of workflow activity. Contains constants for all available types.
16	ActivityValidationModule	Checks for missing activity data (name, caption, ports).
17	AddLabelDialog	Dialogue to add a new label to an activity. Presents a tree with all available label types. Note: Root types cannot be used for labels.
18	<b>AtomicActivity</b>	Workflow activity representing an algorithm, a tool, or a method.
19	AtomicActivityNode	Tree node for atomic workflow activities.
20	AtomicActivityPanel	Extension for basic activity panel. Specialised for atomic activities.
21	BaseActivity	Base implementation of workflow activities. Has common properties such as labels, id, type, caption, description, data ports etc.
22	BaseDataObject	Abstract base class for data objects (e.g. single data object) with common properties such as labels and caption.
23	ChildActivityRepository	Special workflow repository that lists all child activities of a given workflow.
24	CombinedCondition	IfCondition implementation that combined several child conditions using a specified operator (AND, OR).
25	CombinedConditionNode	Tree node for 'AND' or 'OR' multi-conditions.
26	CombinedConditionPanel	Details panel for if-else conditions that implement AND or OR.
27	CombinedRepository	A meta repository combining several workflow repositories.
28	ComparisonCondition	Implementation of if-else condition that compares two values using a specified operator.
29	ComparisonConditionNode	Tree node for conditions that compare two values using a specific operator.
30	ComparisonConditionPanel	Details panel for if-else conditions that compare two values using a specified operator (e.g. >=).
31	CompositeActivityMatcher	Matcher combining multiple sub-matchers.
32	ConditionsRootNode	Root node for if-else condition tree.



#	Class / Interface	Short Description
33	CreateActivityDialog	Dialogue for creating a new workflow activity. Shows a list of available activity types.
34	CreateActivityDialog. ActivityTypeListModel	List model for activity types (such as atomic, loop, ...).
35	CycleDetectionValidationModule	Checks for cycles in acyclic graph activities.
36	DataCardinalityMatchValidationModule	Checks that data sources or forwarded data matches the data cardinality of the target port (single data object vs. data collection).
37	<b>DataCollection</b>	Interface for a collection of data objects. A collection is a data object itself.
38	DataCollectionImpl	Default implementation for data collections. Contains a list of sub-items.
39	DataConversionHelper	Functionality to add a data conversion step for a specific port to a workflow.
40	<b>DataObject</b>	Interface for workflow data objects.
41	DataObjectFactory	Factory for workflow data ports and objects.
42	<b>DataPort</b>	Interface for data ports (input or output).
43	DataPortAlignment	Container for aligning data ports (manual or automatic).
44	DataPortAlignmentPanel	Panel to allow the user to align data ports.
45	DataPortAlignmentPanel. DataPortAlignmentTile	Small panel for one data port. Contains button to open data port details dialogue and up/down buttons to align the ports of the current activity with the ports of the replacement activity.
46	DataPortCorrectionPanel	Validation result panel with data port editing feature.
47	DataPortDialog	Dialogue to create or edit a data port (input port or output port).
48	DataPortImpl	Abstract base implementation for data ports, providing common fields such as data object, id, parent activity.
49	DataPortPanel	Panel representing a single input or output data port of an activity.
50	DataPortPanel.DataObjectPanelListener	Listener interface for data object events (such as removal of a data object).
51	DataRepository	Interface for repositories holding data sources that can be used in workflows.
52	<b>DataTable</b>	Aggregation of data collections which represent table columns.
53	DataTableColumn	Specialisation of an OutputPort. No extended functionality yet.
54	DataTableColumnImpl	Specialised data port implementation for output ports of data tables (representing the columns).
55	DataTableDetails	Panel with input fields for basic properties of a data table.
56	DataTableModel	Implementation of a table model for workflow data tables.

#	Class / Interface	Short Description
57	DataTableTreeNode	Tree node for data tables within the workflow.
58	DataTypeDialog	Dialogue for data type selection from a tree with all available data types.
59	DatatypeMatchValidationModule	Checks that data sources or forwarded data matches the data type of the target port.
60	DefaultDataTable	Default implementation for DataTable.
61	DefaultXmlValidator	Wrapper for XML schema validator
62	DefaultXmlNames (Ontology)	Collection of element and attribute names for XML containing an ontology.
63	DefaultXmlNames (Workflow)	Collection of element and attribute names for XML containing a workflow.
64	DescriptionLabelDialog	Dialogue to manage labels for description objects (that can have labels).
65	DescriptionWithLabels	Special description that can be labelled (for user perspectives).
66	DetailsPanel	Abstract panel class for workflow component details.
67	<b>DirectedGraphActivity</b>	Activity containing a directed acyclic graph (DAG) of child activities.
68	DirectedGraphActivity.ActivityNode	Graph node containing one activity.
69	DirectedGraphActivity. DirectedGraphActivityListener	Listener interface for directed graph activity events
70	DirectedGraphActivityNode	Tree node for 'directed acyclic graph' activities.
71	DirectedGraphActivityPanel	Extension for basic activity panel. Specialised for 'directed acyclic graph' activities.
72	FileFolderRepositoryIndex (now LocalRepositoryIndex)	Index of file folder repositories. Saves and loads the index to the temp directory. Singleton.
73	FileFolderWorkflowRepository	A simple repository implementation that reads all XML workflow files from a specified disk folder.
74	Filter	Common interface for workflow filters.
75	FilterChangeListener	Listener interface for workflow filter change events.
76	FindMatchingActivityDialog	Dialogue to find matching activities from a repository to replace a selected workflow activity or add a child activity.
77	FindMatchingActivityDialog. SimpleMatchValue	Most basic activity match value that has a fixed match score of 100%.
78	<b>ForLoopActivity</b>	Workflow activity representing a for loop.
79	ForLoopActivityNode	Tree node for 'for loop' activities.
80	ForLoopActivityPanel	Extension for basic activity panel. Specialised for 'for loop' activities, showing the loop ports.
81	HasChildActivities	Interface for objects that can have one or more child activities.
82	HasLabels	Interface for objects that can have semantic labels.
83	IdGenerator	Generator for unique IDs (activities, data ports etc.).

#	Class / Interface	Short Description
84	IfCondition	Interface for if-else conditions such as NOT condition or input port condition.
85	IfElseActivity	Activity implementation for conditional branching using if else else ...
86	IfElseActivityNode	Tree node for 'if-else' activities
87	IfElseActivityPanel	Extension for basic activity panel. Specialised for 'if-else' activities.
88	IfElseComparisonPort	Implementation for if-else port mostly delegating to an InputPortImpl object. Used as operand for comparison operations.
89	IfElseConditionPort	Implementation for if-else condition ports mostly delegating to an InputPortImpl object.
90	IfElseConditionTreeNode	Abstract tree node for if-else condition trees.
91	IfElseConditionsTreeModel	Tree model for if-else conditions.
92	IfElsePort	Data port for if-else conditions and comparisons. Extension of InputPort.
93	InputPort	Data input port for workflow activities.
94	InputPortCondition	Condition that evaluates the value of an input port (that can be of type bool or int).
95	InputPortConditionNode	Tree node for if-else conditions that evaluate an input data port.
96	InputPortConditionPanel	Details panel for if-else conditions that evaluate and input port.
97	InputPortImpl	Default implementation of activity input port.
98	<b>Label</b>	A semantic label (e.g. for an activity or a data port). Represents an instance of a label type.
99	LabelFilterElement	Filter element for label filters and matchers.
100	LabelGroup	Group of semantic labels of one type (e.g. a taxonomy).
101	LabelGroup. TooManyLabelsInGroupException	Exception indicating that the label group is full and no more labels can be added.
102	LabellableObjectFilter	Filter for objects that have semantic labels.
103	LabellableObjectMatcher	Semantic label based matcher.
104	LabellableObjectMatcher. LabelMatchValue	MatchValue implementation for Label.
105	LabelListPanel	Container for label panels representing all labels of a workflow activity.
106	LabelMatchValidationModule	Checks that data sources or forwarded data matches the label type of the target port.
107	LabelPanel	Small panel with label root type (heading), label (button), and remove button. This panel is used in the activity details panel (labels of the activity).
108	LabelPanel.LabelPanelListener	Listener for label panel events ('remove label' clicked, ...).
109	Labels	Container for labels (can be used in conjunction with interface 'HasLabels').
110	LabelSelectionTreeRoot	Root node for semantic label type selection tree.

#	Class / Interface	Short Description
111	<b>LabelType</b>	Ontology label type. A type can be abstract (super type for one or more sub-types; equivalent to 'concept'/'class') or it can be concrete (no sub-types; equivalent to 'individual'/'instance').
112	LabelTypesRootTreeNode	Tree node implementation for the ontology tree root.
113	LabelTypeTreeModel	Tree model implementation for ontology label types.
114	LabelTypeTreeNode	Tree node implementation for ontology label types.
115	LabelWorkflowFilter	Workflow filter implementation using labels.
116	LabelWorkflowFilterPanel	Panel with check boxes for all label types of a workflow.
117	LoopHelper	Functionality to add a loop activity to resolve a data cardinality mismatch.
118	LoopPort	Special data port for 'for loop' activities (e.g. for start, end, or step width).
119	LoopPortImpl	Default implementation for LoopPort. Based on default input port implementation.
120	Matcher	Matches a collection of objects against certain criteria and returns a match percentage. Template.
121	MatchValue	Interface template for workflow component matching.
122	MigrationEngine	Migrates the labels of a workflow from an older ontology to the latest one.
123	MigrationMessagesDialog	Dialogue to display ontology / label migration results.
124	MissingAttributePanel	Panel with text edit field to update a text attribute of an activity or a workflow (workflow validation).
125	MissingChildActivitiesValidationModule	Checks if there are any activities that should have a child activity but do not have one.
126	MissingDataTypePanel	Panel with data type selection control a data port of an activity (workflow validation).
127	MissingDatatypesValidationModule	Checks whether all data ports have a data type specified. Ignores abstract activities.
128	NotCondition	Implements NOT operator for a child if-else condition.
129	NotConditionNode	Tree node for 'NOT' if-else condition.
130	NotConditionPanel	Details panel for NOT if-else condition.
131	<b>Ontology</b>	Ontology of labels for activities and data ports. Singleton.
132	Ontology.DefaultRootType	Access to default root label types. Deprecated.
133	OntologyEditor	Graphical editor for label type hierarchies and label slots of activities/data objects.
134	OntologyInitialiser	Interface for classes that initialise an ontology with label type hierarchies (taxonomies).

#	Class / Interface	Short Description
135	OntologyMigrationRulesDialog	Dialogue that allows the definition of rules for migration from an older ontology to the current one.
136	OntologyWriter	Interface for writers that save an ontology to a file.
137	OtherOntology	Wrapper for ontology class to allow the instantiation of multiple ontologies (the base class is a singleton).
138	OutputPort	Interface for data output ports for workflow activities. Specialises DataPort.
139	OutputPortImpl	Default implementation for activity output ports.
140	OwlOntologyWriter	XML writer for ontology label types. Uses OWL XML format.
141	OwlWorkflowSemanticsWriter	XML writer for semantic information of workflows. Uses OWL XML format.
142	OwlXmlNames	Element and attribute names for OWL XML format.
143	Pair	Simple pair of objects (left and right).
144	PortLinkTreeNode	Tree node for data port link selection (source or forwarding).
145	ProblemResolutionDescriptionPanel	Simple text panel that shows possible user actions to resolve a validation problem.
146	RecentDocuments	Collection of recent documents (workflows) that can be saved to / loaded from the temp dir.
147	ReplaceActivityDialog	Dialogue to choose what activity details to copy from a current activity to a replacement activity.
148	RepositoryHub	Manager for workflow repositories <ul style="list-style-type: none"> <li>• Show registered repositories</li> <li>• Add/remove repositories</li> <li>• Show all workflows of a repository</li> <li>• Add, open, edit, remove workflows</li> </ul>
149	SelectPortLinkDialog	Dialogue for selecting a source of an input port or a 'forwarded from' for output ports.
150	SimpleOntologyWriter	XML writer for ontology label types. Uses a custom XML format.
151	SingleDataObject	Data object implementation that represents a single piece of data such as a file, an integer, or a text string.
152	TableContentDialog	Dialogue for workflow data table content. Shows an editable table view.
153	TextSearchWorkflowFilter	Filter that looks for a search string with workflow and root activity meta data.
154	UnconnectedDataPortsValidationModule	Checks for 'loose ends' (input ports that have no source or output ports that are not forwarded anywhere).
155	User	Interface for workflow system users.
156	UserImpl	Default implementation for the 'User' interface.
157	UserManagement	Singleton for user-related functionality.

#	Class / Interface	Short Description
158	ValidationResultTreeItem	Tree item specialisation for workflow validation result items (errors, warnings etc.).
159	ValidationTreeCellRenderer	Specialised tree renderer that adds icons for validation errors, warnings and notes.
160	WelcomePanel	Placeholder panel for workflow component details that is displayed when the editor is opened.
161	<b>Workflow</b>	Wrapper for an activity (root activity) with metadata and optional sub-workflows.
162	WorkflowConcretisationDialog	Dialogue for making an abstract workflow concrete (assisted or automated).
163	WorkflowConcretisationInteractionDialog	Dialogue for user interaction during workflow concretisation.
164	WorkflowConcretisationManager	Interface for concretisation events, possibly requiring user interaction.
165	WorkflowConcretiser	Tries to make an abstract workflow concrete by replacing all abstract atomic activities with concrete ones from a activity repository.
166	WorkflowDataType	Data type for workflow activity data ports. Has a parent and children, creating a tree structure.
167	WorkflowDataTypeHierarchy	Singleton containing all predefined data types that can be used in activities.
168	WorkflowDataTypeHierarchy. DataTypeSaxHandler	SAX handler implementation to parse data type hierarchies.
169	WorkflowDetails	Panel with fields for workflow attributes such as name, author and version. Also offers an entry point for workflow concretisation.
170	WorkflowEditor	Main class for graphical workflow editor (Swing user interface).
171	WorkflowFilter	Interface for filters that can be applied to an workflow list.
172	WorkflowImpl	Default implementation for the Workflow interface.
173	WorkflowObjectValidationModule	Workflow validation module that checks the main workflow object itself.
174	<b>WorkflowRepository</b>	Interface for repositories holding workflows.
175	WorkflowRepositoryLabelExporter	Export of all used labels and their counts from all workflows of a repository. Table format: Workflow, [label type 1], [label type 2], ... aletheia, 1, 0, ...
176	WorkflowRootNode	Root node for workflow and activity tree.
177	WorkflowSearchResultGrid	Grid view for workflow search result items. Scrollable.
178	WorkflowSearchResultItemPanel	Panel for a single workflow search result item. The item is clickable and the respective action can be handled with a listener.
179	WorkflowSearchResultView	View interface for workflow search results from a repository.

#	Class / Interface	Short Description
180	WorkflowSearchTool	Dialogue to search workflow repository using label filters.
181	WorkflowTreeModel	Tree model for workflows (includes workflow root and all activities).
182	WorkflowTreeNode	Abstract tree node for workflow trees (workflow root or activity node).
183	WorkflowUmlDialog	Experimental dialogue with UML view of workflow (activity diagram).
184	WorkflowUmlView	Experimental UML renderer for a workflow (activity diagram).
185	WorkflowValidationDialog	Validation of workflow and presentation of results (errors, warnings and notes).
186	WorkflowValidationModule	Interface for workflow validation module that checks a single issue.
187	WorkflowValidationResult	Result item of workflow validation. Can have nested items, creating a tree structure.
188	<b>WorkflowValidator</b>	Validation of workflows. Contains a set of validation modules.
189	XmlLabelTypeHierarchyInitialiser	Loads the label type hierarchies (taxonomies) for an ontology from an XML file.
190	XmlLabelTypeHierarchyInitialiser. OntologySaxHandler	SAX handler implementation to parse label type hierarchies.
191	XmlWorkflowReader	Reader for workflow XML files (SAX).
192	XmlWorkflowReader. WorkflowSaxHandler	SAX handler implementation to parse a workflow XML file.
193	XmlWorkflowWriter	XML writer for workflows (DOM).

## Appendix C – Software tools used for activity repository

#	Tool	Description
1	abbot	<p>Abbot is a tool for undertaking large-scale conversion of XML document collections in order to make them interoperable with one another. In particular, Abbot can make one or more collections conform to a designated schema (including a schema used to define one of the collections). In the simplest case, where the "target schema" is a proper subset of the schema(s) used to define the collections in question, Abbot operates more-or-less automatically. More complicated cases require that you define specific transformations in a configuration file, but that configuration file uses a simple language unrelated to either XSLT or Schema.</p> <p>By default, Abbot converts documents into TEI Analytics -- a TEI subset designed for text analysis applications.</p> <p><a href="https://github.com/CDRH/abbot">https://github.com/CDRH/abbot</a></p>
2	ABBYY Business Card Reader	<p>ABBYY Business Card Reader 2.0 (for Windows) automatically digitizes paper business cards data and transforms it into systemized contact database. This software designed to help professionals boost their productivity, raise the effectiveness of their business communications, save time and eliminate tedious retyping. The application works perfect with virtually any scanning device.</p> <p><a href="https://www.abbyy.com/business-card-reader-for-windows/">https://www.abbyy.com/business-card-reader-for-windows/</a></p>
3	ABBYY FineReader Engine 11	<p>ABBYY FineReader Engine lets developers, integrators and BPOs integrate optical text recognition technologies into their applications. Based on the ABBYY recognition platform, this SDK enables scanned documents and images to be transformed into searchable and editable document formats – and delivers award-winning OCR, intelligent character recognition (ICR), barcode, checkmark and field-level/zonal recognition and PDF conversion.</p> <p><a href="https://www.abbyy.com/ocr-sdk/">https://www.abbyy.com/ocr-sdk/</a></p>
4	ABBYY Image Pre-processing	<p>"[...] a set of image pre-processing tools and allows you to watch how this or that tool influences recognition quality. You can use general pre-processing tools (like page orientation and skew correction), filter colors, use special pre-processing tools for photos, and enhance appearance of the images."</p> <p><a href="http://help.abbyy.com/FineReader/FineReader12/English/GettingImage/AdjustImage.htm">http://help.abbyy.com/FineReader/FineReader12/English/GettingImage/AdjustImage.htm</a></p>
5	ABBYY Binarisation	<p>Prior to analysing the structure of the document and identifying its blocks, an OCR program will binarize the image, i.e.: it will convert a colour or a greyscale image into a monochrome one (1 bit). Modern documents will often include complex design elements as textures and background images. If an OCR program would use only a very simple binarisation there will be too many excess dots left around the characters, which will have an adverse effect on the quality of recognition.</p> <p>The same is true about binarising background images. Therefore, it is crucial that the program can separate the text from the underlying textures and images. To solve this issue ABBYY technologies use two pre-processing procedures Intelligent Background Filtering and Adaptive Binarization.</p> <p><a href="https://abbyy.technology/en:features:ocr:adaptive_binarisation">https://abbyy.technology/en:features:ocr:adaptive_binarisation</a></p>
6	ABBYY Block Segmentation	<p>Before characters and words can be recognised by an OCR engine, the print space of the image has to be identified, and from there paragraphs and lines. This tool can be used to identify blocks on a scanned document.</p> <p><a href="http://www.digitisation.eu/tools-resources/tools-for-text-digitisation/abbyy-block-segmentation/">http://www.digitisation.eu/tools-resources/tools-for-text-digitisation/abbyy-block-segmentation/</a></p>
7	AGORA	<p>Analysis and indexation of Historical and Degraded Documents: pre-processing, layout analysis and character recognition.</p> <p>In this paper, we describe how meta-data of indexation can be extracted from historical document images using an interactive process with a software called AGORA. The algorithms involved in AGORA use two maps to segment noisy images: a shape map that focuses on connected components and a background map that provides information on white areas corresponding to block separations in</p>



		the page. Using a first segmentation result obtained by using these two maps, meta-data can be extracted according to scenarios produced by the users. These scenarios are defined very simply during an interactive stage. The user is able to make processing sequences adapted to the different kinds of images he is likely to meet and according to the desired meta-data. Finally, we describe different experimentations that have been done during the BVH project to test the usability and the performances of AGORA software. [204]
8	Aletheia	<p>Aletheia is an advanced system for accurate and yet cost-effective analysis, recognition and annotation of scanned documents. It aids the user with a number of automated and semi-automated tools which were developed and fine-tuned based on feedback from major libraries across Europe and from their digitisation service providers which are using it in a production environment.</p> <p>Cutting-edge features are, among others, the support of top-down ground truthing with sophisticated split and shrink tools as well as bottom-up ground truthing supporting the aggregation of lower-level elements to more complex structures. The integrated rules and guidelines validator, in combination with powerful correction tools, enable efficient production of highly accurate ground truth as well as standardised electronic renditions of digitised documents.</p> <p>In addition, special features such as a customisable virtual keyboard and the Aletheia Sans font with extensive coverage of special characters in Unicode have been developed to support working with the complexities of historical documents.</p> <p><a href="http://www.primaresearch.org/tools/Aletheia">http://www.primaresearch.org/tools/Aletheia</a></p>
9	Aletheia Dewarping Ground Truth Creation	<p>Dewarping is a grid-based method for geometric correction of document images. Aletheia allows creating and saving dewarping ground truth as well as load existing dewarping data from XML files. Several grids (non-overlapping) can be defined for one page. Aletheia also allows working on two independent sets of grids in parallel (e.g. for comparing ground truth against grid detection result). A dewarping grid is a matrix of points that are connected to each other horizontally and vertically, forming grid lines.</p> <p>In addition, reference lines can be defined. By default, the reference lines are position at the average location of all corresponding grid points.</p>
10	ALTO-Edit	<p>ALTO Editor for text and segmentation. Browser based post correction tool for Alto XML files.</p> <p><a href="https://github.com/KBNLresearch/alto-editor">https://github.com/KBNLresearch/alto-editor</a></p>
11	Apache OpenNLP Named Entity Recognition	<p>The Apache OpenNLP library is a machine learning based toolkit for the processing of natural language text.</p> <p>It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. These tasks are usually required to build more advanced text processing services. OpenNLP also includes maximum entropy and perceptron based machine learning.</p> <p>The Name Finder can detect named entities and numbers in text. To be able to detect entities the Name Finder needs a model. The model is dependent on the language and entity type it was trained for. The OpenNLP projects offers a number of pre-trained name finder models which are trained on various freely available corpora. They can be downloaded at our model download page. To find names in raw text the text must be segmented into tokens and sentences.</p> <p><a href="https://opennlp.apache.org/">https://opennlp.apache.org/</a></p>
12	Apache OpenNLP Sentence Detector	<p>The OpenNLP Sentence Detector can detect that a punctuation character marks the end of a sentence or not. In this sense a sentence is defined as the longest white space trimmed character sequence between two punctuation marks. The first and last sentence make an exception to this rule. The first non-whitespace character is assumed to be the begin of a sentence, and the last non whitespace character is assumed to be a sentence end.</p>
13	Apache OpenNLP Tokenizer	<p>The OpenNLP (Natural Language Processing) Tokenizers segment an input character sequence into tokens. Tokens are usually words, punctuation, numbers, etc.</p>
14	ASV Toolbox - JLanI	<p>The JLanI tool allows you to identify a language on the basis of a text input. This input can be given by a file, database connection or plain text.</p> <p>Per language, a list of words and their frequency is needed for training. This list can come from a database or a text file.</p> <p>This tool use a file called blacklist.txt which you find in resources/jlani. It should be an empty file if you download this tool. JlanI needs this file even if it is empty (so do not delete it) but you can fill the file with words which should be not used for language identification. It contains one word per line. For example, you should write name like George, Bush, Angelika, Merkel or USA, Deutschland and abbreviations like A. or G. or W. in this file. This will improve the result in case you only enter words not specific to any language.</p>

		<a href="http://wortschatz.uni-leipzig.de/~cbiemann/software/toolbox/">http://wortschatz.uni-leipzig.de/~cbiemann/software/toolbox/</a>
15	AWE Layout Editor	Aletheia Web Edition crowd sourcing post-processing tool for page layout and text content. Also known as AletheiaWeb.  <a href="http://www.primaresearch.org/tools">http://www.primaresearch.org/tools</a>
16	Berkeley Parser	Parsing is the task of analyzing the grammatical structure of natural language. Given a sequence of words, a parser forms units like subject, verb, object and determines the relations between these units according to some grammar formalism. Our work has focused on learning probabilistic context-free grammars (PCFGs) which assign a sequence of words the most likely parse tree. The parser supports a variety of languages and achieves state-of-the-art performance on most of them.  <a href="https://github.com/slavpetrov/berkeleyparser">https://github.com/slavpetrov/berkeleyparser</a>
17	Brevity	Businesses and other organizations often deal with hundreds or even hundreds of thousands of documents. Knowing the content of these documents can be difficult. While you can discern the content of a graphical image at a glance, with text documents you have to read through each to discern it's content. Reading through an entire document takes time - time you do not have to waste. The traditional solution to this problem has been to assign people to read the documents and write a brief abstract for each one. Unfortunately many organizations simply do not have the resources to assign people to summarize hundreds or even thousands of documents. Brevity provides you with a solution. Brevity easily generates document summaries for you. The summaries can be as long or as short as you wish. You can also use Brevity to highlight key sentences or words in your document.  Key Benefits of Brevity: Accurately generate automated document summaries Allow users to quickly determine a document's contents at a glance Highlight significant words and sentences in a document Find the key parts of a document  <a href="http://www.lextek.com/brevity/">http://www.lextek.com/brevity/</a>
18	CamScanner	CamScanner helps you scan, store, sync and collaborate on various contents across smartphones, tablets and computers. Features: <ul style="list-style-type: none"> <li>- Mobile Scanner: Use your phone camera to scan receipts, notes, invoices, whiteboard discussions, business cards, certificates, etc.</li> <li>- Optimize Scan Quality: Smart cropping and auto enhancing make the texts and graphics look clear and sharp</li> <li>- Quick Search: By entering any keyword, you'll see a list of docs with the word in their titles, notes or images (Registrants only)</li> <li>- Extract Texts from Image: OCR (optical character recognition) extracts texts inside images for further editing or .txt sharing. (Paid app only)</li> </ul> <a href="https://www.camscanner.com/">https://www.camscanner.com/</a>
19	Clara OCR	Clara OCR is an Optical Character Recognition program. It features both a powerful GUI for the X Window System, and a Web interface. The Web interface is able to collect revision efforts from the Internet, using a simple revision model. It is intended to be used in the cooperative optical recognition of old books. It tries to facilitate fine-tuning, so an optical recognition project is enabled to invest resources in tuning the OCR, in order to achieve better recognition results for one specific book, and reduce the overall revision cost.  <a href="http://gnu.gds.tuwien.ac.at/directory/claraocr.html">http://gnu.gds.tuwien.ac.at/directory/claraocr.html</a>
20	CLAWS	Part-of-speech (POS) tagging, also called grammatical tagging, is the commonest form of corpus annotation, and was the first form of annotation to be developed by UCREL at Lancaster. Our POS tagging software for English text, CLAWS (the Constituent Likelihood Automatic Word-tagging System), has been continuously developed since the early 1980s. The latest version of the tagger, CLAWS4, was used to POS tag c.100 million words of the British National Corpus (BNC).  <a href="http://ucrel.lancs.ac.uk/claws/">http://ucrel.lancs.ac.uk/claws/</a>
21	cue.language	cue.language is a small library of Java code and resources that provides the following basic natural-language processing capabilities:

		<ul style="list-style-type: none"> <li>- Tokenizing natural language text into individual words</li> <li>- Tokenizing natural language text into sentences</li> <li>- Tokenizing natural language text into n-grams (sequences of 2 or more words that appear next to each other in a sentence)</li> <li>- Counting strings</li> <li>- Detecting which script (alphabet, writing system) is required to represent a text</li> <li>- Guessing what language a text is in</li> <li>- Customizable "stop word" detection for a variety of languages</li> </ul> <p><a href="https://github.com/vcl-xx/cue.language">https://github.com/vcl-xx/cue.language</a></p>
22	Document capturing by camera	Photographing paper documents to acquire digital images. A generic activity for creating digital copies using a camera.
23	Document scanning	Scanning of paper documents to acquire digital images. A generic activity for creating digital copies using a scanner.
24	File Analyzer	<p>The NARA File Analyzer Tool walks a directory tree and performs a "File Test" on each file that is encountered. The application framework allows new File Tests to be quickly developed and deployed into the application. The results of each File Test are compiled into a table that summarizes the results of the analysis.</p> <p>A File Test is a simple set of actions that are performed upon a single file such as filename validation, file size statistical analysis, checksum calculation, file type extraction. Depending on the action, the content of the file may or may not be read. Each File Test is configured with filters that determine which files will be processed by the File Test (i.e. only image files).</p> <p>Each File Test will generate a table of results. The number of columns and the definition of the columns will vary from test to test. For example, a file type analysis will report the file extension and the number of files discovered with that extension. The checksum file tests will report the name of a file and the checksum string associated with that file.</p> <p>The File Analyzer tool can be run as a GUI in which the results are displayed in a table. The File Analyzer can also be run in batch mode. In batch mode, the results will be written to a tab-separated file. The GUI version of the application allows the results of multiple executions to be merged. The merged information can be filtered to display matching values and mismatched values.</p> <p><a href="https://github.com/usnationalarchives/File-Analyzer">https://github.com/usnationalarchives/File-Analyzer</a></p>
25	Document Deskewer	<p>Generic skew detection and correction (for the full range 0-360 degrees) for documents printed using Roman scripts.</p> <p><a href="http://www.digitisation.eu/training/succeed-training-materials/image-processing/document-deskewer/">http://www.digitisation.eu/training/succeed-training-materials/image-processing/document-deskewer/</a></p>
26	Fraunhofer Newspaper Segmenter	<p>Award-winning (e.g. ICDAR'09,'11) page and article segmentation for scanned documents featuring complex layouts (e.g. (historical) newspapers, contemporary magazines, text books, etc.)</p> <p><a href="http://www.iais.fraunhofer/">http://www.iais.fraunhofer/</a></p>
27	FromThePage	<p>FromThePage is an open-source tool that allows volunteers to collaborate to transcribe handwritten documents.</p> <p>Features:</p> <p>Wiki-style Editing: Users add or edit transcriptions using simple, wiki-style syntax on one side of the screen while viewing a scanned image of the manuscript page on the other side.</p> <p>Version Control: Changes to each page transcription are recorded and may be viewed to follow the edit history of a page.</p> <p>Wikilinks: Subjects mentioned within the document may be indexed via simple wikilinks within the transcription. Users can annotate subjects with full subject articles.</p> <p>Presentation: Readers can view transcriptions in a multi-page format or alongside page images. They can also read all the pages that mention a subject</p> <p>Automatic Markup: FromThePage can suggest wikilinks to editors by mining previously edited transcriptions. This helps insure editorial consistency and vastly reduces the amount of effort involved in markup.</p> <p>Internet Archive integration: FromThePage can be pointed at manuscripts hosted on Archive.org. It will import the page structure and any printed page titles into its native format for transcription, while serving page images from the Internet Archive.</p> <p><a href="http://beta.fromthepage.com/">http://beta.fromthepage.com/</a></p>

28	Functional Extension Parser	<p>The Functional Extension Parser (FEP) is a Document Understanding Software tool capable of decoding layout elements of books.</p> <p>Based on the output of Optical Character Recognition, layout elements such as page numbers, running titles, headings, and footnotes are detected and annotated. The FEP has been trained and tested on several datasets comprising books from the 18th to the 20th century from several European libraries. The final release is a production tool ready to be used for the structural annotation of digitised documents.</p> <p><a href="https://impactocr.wordpress.com/2010/05/07/the-functional-extension-parser-a-rule-based-system-for-flexible-structural-analysis/">https://impactocr.wordpress.com/2010/05/07/the-functional-extension-parser-a-rule-based-system-for-flexible-structural-analysis/</a></p>
29	GEDI	<p>GEDI (Groundtruthing Environment for Document Images) is a highly configurable document image annotation tool. Its basic structure involves two types of files, an Image file, and a corresponding .XML</p> <p>Features</p> <ul style="list-style-type: none"> <li>- GEDI listener: the interface can be controlled by programs outside of java; it accepts specific HTTP commands.</li> <li>- Data overlay: visualization of multiple metadata file at the same time.</li> <li>- Commenting and tracking: users can annotate zones with questions and issues.</li> <li>- Scripting: to process the metadata on an image through a configurable script.</li> <li>- Connected components and RL encoding: it generates connected components and/or underlying run length encoding.</li> <li>- Able to work with color images; able to convert to Black and White</li> </ul> <p><a href="https://sourceforge.net/projects/gedigroundtruth/">https://sourceforge.net/projects/gedigroundtruth/</a></p>
30	GIMP	<p>GIMP Image Processing Tool</p> <p><a href="https://www.gimp.org/">https://www.gimp.org/</a></p>
31	Goggles	<p>Google Goggles is an image recognition mobile app developed by Google.[1] It is used for searches based on pictures taken by handheld devices. For example, taking a picture of a famous landmark searches for information about it, or taking a picture of a product's barcode searches for information on the product.</p> <p>Google Goggles works better with certain types of queries. It can recognize up to three items at a time. For best results, try taking pictures of the following: Books &amp; DVDs, Landmarks, Barcodes &amp; QR codes, Logos, Contact info, Artwork, Businesses, Products, Text</p> <p><a href="https://en.wikipedia.org/wiki/Google_Goggles">https://en.wikipedia.org/wiki/Google_Goggles</a></p>
32	Graph-based Dependency Parser (mate-tools)	<p>The tools provide a pipeline of modules that carry out lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence. The system's two main components draw on improved versions of a state-of-the-art dependency parser (Bohnet, 2010) and semantic role labeler (Björkelund et al., 2009) developed independently by the authors. The tools are language independent, provide a very high accuracy and are fast. The dependency parser had the top score for German and English dependency parsing in the CoNLL shared task 2009.</p> <p><a href="https://code.google.com/archive/p/mate-tools/">https://code.google.com/archive/p/mate-tools/</a></p>
33	Ground Truth Maker	<p>An application based on the lexicon defined by historians NaviDoMass. This application allows you to create the ground truth associated with an image to test the different tools developed by computer scientists.</p> <p>The software is only available in Windows Installer Version, one with .NET Framework 3.5 and one without for those who already have</p> <p><a href="http://navidomass.univ-lr.fr/gtm.html">http://navidomass.univ-lr.fr/gtm.html</a></p>
34	GTText	<p>OCR free software and Ground Truthing tool for Color Images with Text.</p> <p>In the research of algorithms that extract text from color images a set of files with the exact location of the text is needed to avoid inefficient and tedious visual checks of the results.</p> <p>This Ground Truth information saves enormous time and gives accuracy. For this the gttxt project helps to create fast and quality Ground Truthed data-sets from color text images.</p> <p>It performs fast OCR text recognition and copies image text to clipboard. Loads from screen snapshots, image files and scans documents.</p> <p>The basic level of work is at a pixel detection, making possible to group regions to form the glyph or even use a direct editing to get the choice.</p>

		<p>Extracting text from complex color images can be done immediately just by selecting the region of the open, scanned or pasted image document.</p> <p><a href="https://code.google.com/p/gttext/">https://code.google.com/p/gttext/</a></p>
35	Hectography Foreground Extractor	<p>Hectography Foreground Extractor (The hectograph or gelatin duplicator or jellygraph is a printing process which involves transfer of an original, prepared with special inks, to a pan of gelatin or a gelatin pad pulled tight on a metal frame.)</p> <p>Foreground-background separation in color (3-channel) scans of hectographic copies, allowing an order of magnitude improvement in OCR quality.</p> <p><a href="http://www.iais.fraunhofer/">http://www.iais.fraunhofer/</a></p>
36	Hot Metal Font Enhancer	<p>Fraunhofer Hot Metal Font Enhancer (In printing and typography, hot metal typesetting refers to technologies for typesetting text in letterpress printing. This method injects molten type metal into a mold that has the shape of one or more glyphs. The resulting sorts and slugs are later used to press ink onto paper.)</p> <p>Font enhancement of prints produced hot metal typesetting allowing higher OCR accuracy.</p> <p><a href="http://www.iais.fraunhofer/">http://www.iais.fraunhofer/</a></p>
37	HOcr Eval	<p>Evaluate the actual OCR with respect to the ground truth. This outputs the number of OCR errors due to incorrect segmentation and the number of OCR errors due to character recognition errors.</p> <p>It works by aligning segmentation components geometrically, and for each segmentation component that can be aligned, computing the string edit distance of the text the segmentation component contains.</p> <p>hOCR is a format for representing OCR output, including layout information, character confidences, bounding boxes, and style information. It embeds this information invisibly in standard HTML.</p> <p><a href="https://github.com/tmbdev/hocr-tools">https://github.com/tmbdev/hocr-tools</a></p>
38	ImageMagick	<p>ImageMagick is a software suite to create, edit, compose, or convert bitmap images.</p> <p><a href="http://www.imagemagick.org/">http://www.imagemagick.org/</a></p>
39	IMPACT Spelling Variation Tool	<p>The IMPACT Spelling Variation Tool deals with historical spelling variation. It provides functionality to estimate a model of spelling variation from example data, and to match a historical word, or a list of historical words, to a list of 'modern' words (or historical words in normalized, modern-like spelling).</p>
40	Islandora	<p>A javascript based tei editor. BETA</p> <p><a href="https://github.com/islandora-deprecated/islandora_tei_editor">https://github.com/islandora-deprecated/islandora_tei_editor</a></p>
41	ISRI Evaluation Tool	<p>This tool evaluates the performance of an optical character recognition system on character and word level.</p> <p>Source code of OCR evaluation tools used in the UNLV/ISRI annual tests of OCR Accuracy.</p> <p><a href="https://github.com/eddieantonio/isri-ocr-evaluation-tools">https://github.com/eddieantonio/isri-ocr-evaluation-tools</a></p>
42	jMet2Ont	<p>jMet2Ont is a mapping tool that transforms XML-based metadata to ontology-based formats. The source metadata format may be flat (e.g. Dublin Core) or hierarchical (e.g. MARC/XML).</p> <p>The mapping rules are described in an XML file. To use the mapper you do not need any programming knowledge. You can find full specification of the XML rule syntax in the Documentation section.</p> <p><a href="http://www.carpet-project.net/en/catalogue/detail/jmet2ont/">http://www.carpet-project.net/en/catalogue/detail/jmet2ont/</a></p>
43	Layout Analysis Performance Visualisation	<p>PRImA Layout Analysis Page analysis and recognition performance visualisation using PRImA Layout Evaluation Tool.</p> <p><a href="http://www.primaresearch.org/tools/PerformanceEvaluation">http://www.primaresearch.org/tools/PerformanceEvaluation</a></p>
44	LayoutEval	<p>The PRImA Layout Evaluation Tool is part of a framework for evaluating the performance of layout analysis methods. It combines efficiency and accuracy by using a special interval based geometric representation of regions. A wide range of sophisticated evaluation measures provides the means for a deep insight into the analysed systems, which goes far beyond simple benchmarking. The support of user-defined profiles allows the tuning for practically any kind of evaluation scenario related to real world applications.</p>

		<p>The framework has been successfully delivered as part of a major EU-funded project (IMPACT) to evaluate large-scale digitisation projects and has been validated applied within the past three ICDAR Page Segmentation Competitions.</p> <p><a href="http://www.primaresearch.org/tools/PerformanceEvaluation">http://www.primaresearch.org/tools/PerformanceEvaluation</a></p>
45	Lios	<p>Lios (Linux-intelligent-ocr-solution) is a free and open source software for converting print into text using either scanner or a camera. It can also produce text out of scanned images from other sources such as pdfs, images or folders containing images.</p> <p><a href="https://sourceforge.net/projects/lios/">https://sourceforge.net/projects/lios/</a></p>
46	MALLET Document Classification	<p>Classify text document according to trained model. MALLET provides a simple interface to a large collection of classification algorithms.</p> <p><a href="http://mallet.cs.umass.edu/classifier-devel.php">http://mallet.cs.umass.edu/classifier-devel.php</a></p>
47	MapForce	<p>Altova MapForce® 2013 is an award-winning any-to-any graphical data mapping, conversion, and integration tool that maps data between any combination of XML, database, flat file, EDI, Excel, XBRL, and/or Web service, then transforms data instantly or autogenerates royalty-free data integration code for the execution of recurrent conversions.</p> <p><a href="http://www.altova.com/mapforce.html">http://www.altova.com/mapforce.html</a></p>
48	Metadata Extraction Tool	<p>The Metadata Extraction Tool was developed by the National Library of New Zealand to programmatically extract preservation metadata from a range of file formats like PDF documents, image files, sound files Microsoft office documents, and many others. The Tool builds on the Library's work on digital preservation, and its logical preservation metadata schema. It is designed to:</p> <ul style="list-style-type: none"> <li>- automatically extracts preservation-related metadata from digital files</li> <li>- output that metadata in a standard format (XML) for use in preservation activities.</li> </ul> <p>The Tool was designed for preservation processes and activities, but can be used to for other tasks, such as the extraction of metadata for resource discovery.</p> <p>Supported File Formats Images: BMP, GIF, JPEG and TIFF. Office documents: MS Word (version 2, 6), Word Perfect, Open Office (version 1), MS Works, MS Excel, MS PowerPoint, and PDF. Audio and Video: WAV, MP3 (normal and with ID3Tags), BFW, FLAC. Markup languages: HTML and XML. Internet files: ARC</p> <p><a href="http://meta-extractor.sourceforge.net/">http://meta-extractor.sourceforge.net/</a></p>
49	MILE OCR Performance Evaluator	<p>A desktop application used for performance evaluation of Optical Character Recognizers (OCR). Implemented using Eclipse SWT and runs on Windows &amp; Linux.</p> <p><a href="http://www.findbestopensource.com/product/ocr-performance-evaluator">http://www.findbestopensource.com/product/ocr-performance-evaluator</a></p>
50	MontyChunker	<ul style="list-style-type: none"> <li>- chunks tagged text into verb, noun, and adjective chunks (VX,NX, and AX respectively)</li> <li>- incredible speed and accuracy improvement over previous MontyChunker</li> </ul> <p><a href="http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html">http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html</a></p>
51	MontyLemmasiser	<p>Strips inflectional morphology, i.e. changes verbs to infinitive form and nouns to singular form</p> <ul style="list-style-type: none"> <li>- part-of-speech sensitive lemmatisation</li> <li>- strips plurals (geese--&gt;goose) and tense (were--&gt;be, had--&gt;have)</li> <li>- includes regexps from Humphreys and Carroll's morph.lex, and UPENN's XTAG corpus</li> </ul> <p><a href="http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html">http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html</a></p>
52	MontyTokenizer	<p>Tokenizes raw English text (sensitive to abbreviations), and resolve contractions, e.g. "you're" ==&gt; "you are"</p> <p><a href="http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html">http://alumni.media.mit.edu/~hugo/montylingua/doc/MontyLingua.html</a></p>
53	NCSR Binarisation	<p>Performs image binarisation using an algorithm developed at NCSR.</p>

		<a href="http://www.demokritos.gr/">http://www.demokritos.gr/</a>
54	NCSR Border Removal	This tool detects and removes noisy black borders as well as noisy text regions. Moreover, it detects the optimal page frames of double page document images.  <a href="http://www.demokritos.gr/">http://www.demokritos.gr/</a>
55	NCSR Character Segmentation	The developed methodology takes as input isolated words and separates them into characters.  <a href="http://www.demokritos.gr/">http://www.demokritos.gr/</a>
56	NCSR OCR Evaluation Tool	Evaluation Tool for OCR by National Center for Scientific Research (NCSR) "Demokritos"  <a href="http://www.demokritos.gr/">http://www.demokritos.gr/</a>
57	NCSR Page Curl Correction	This tool rectifies document images which suffer from warping and perspective distortions that deteriorate the performance of current OCR approaches.  <a href="http://www.demokritos.gr/">http://www.demokritos.gr/</a>
58	ocrad	GNU Ocrad is an OCR (Optical Character Recognition) program based on a feature extraction method. It reads images in pbm (bitmap), pgm (greyscale) or ppm (color) formats and produces text in byte (8-bit) or UTF-8 formats. Also includes a layout analyser able to separate the columns or blocks of text normally found on printed pages. Ocrad can be used as a stand-alone console application, or as a backend to other programs.  <a href="https://www.gnu.org/software/ocrad/">https://www.gnu.org/software/ocrad/</a>
59	OCROpus OCR	OCROpus™ is an OCR system written in Python, NumPy, and SciPy focusing on the use of large scale machine learning for addressing problems in document analysis.  <a href="https://github.com/tmbdev/ocropy">https://github.com/tmbdev/ocropy</a>
60	OneNote Handwriting Recognition	Microsoft OneNote handwriting recognition feature  <a href="https://support.office.com/en-us/article/Take-notes-in-your-own-handwriting-46FE3AB9-5747-4820-B646-0C08935C5991">https://support.office.com/en-us/article/Take-notes-in-your-own-handwriting-46FE3AB9-5747-4820-B646-0C08935C5991</a>
61	Otsu Binarisation	Otsu parameter free image binarisation using a global threshold (part of PRIMa Image Tool)  <a href="http://www.primaresearch.org/tools">http://www.primaresearch.org/tools</a>
62	Pandoc	If you need to convert files from one markup format into another. Pandoc can convert documents in markdown, reStructuredText, textile, HTML, DocBook, LaTeX, MediaWiki markup, OPML, Emacs Org-Mode, Txt2Tags, Microsoft Word docx, EPUB, or Haddock markup to HTML formats, Word processor formats, Ebooks, Documentation formats, Page layout formats, ... Pandoc understands a number of useful markdown syntax extensions, including document metadata (title, author, date); footnotes; tables; definition lists; superscript and subscript; strikeout; enhanced ordered lists (start number and numbering style are significant); running example lists; delimited code blocks with syntax highlighting; smart quotes, dashes, and ellipses; markdown inside HTML blocks; and inline LaTeX.  <a href="http://pandoc.org/">http://pandoc.org/</a>
63	PDF-XChange Viewer	The No.1 rated BEST PDF Reader  <a href="http://www.tracker-software.com/product/pdf-xchange-viewer">http://www.tracker-software.com/product/pdf-xchange-viewer</a>
64	PRImA Crowd Prototype	Crowd sourcing prototype for correcting OCRed text on text line level.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
65	PRImA Dewarping Evaluation	Grid-based evaluation of document image dewarping data.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
66	PRImA Dewarping	Correction of arbitrary warping effects in document images (e.g. due to humidity). There are two versions: a command line tool which can be used in automated workflows and a GUI tool allowing manual intervention, including showcase production as well as ground truth creation.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
67	PRImA Glyph Extraction	Extracts glyph image snippets from a document page scan using glyphs defined in PAGE XML

		<a href="http://www.primaresearch.org/tools/ExtractorExporter">http://www.primaresearch.org/tools/ExtractorExporter</a>
68	PRImA JFeatureExtractor	Extracts text features from page content (PAGE XML), for usage in quality estimation.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
69	PRImA JPageViewer	Simple viewer for PAGE XML files (layout + text content).  <a href="http://www.primaresearch.org/tools/PAGEViewer">http://www.primaresearch.org/tools/PAGEViewer</a>
70	PRImA Layout Aligner	Page layout alignment via template matching  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
71	PRImA LayoutEval Profile Creation	User interface to create evaluation profile for PRImA LayoutEval.  <a href="http://www.primaresearch.org/tools/PerformanceEvaluation">http://www.primaresearch.org/tools/PerformanceEvaluation</a>
72	PRImA Page Feature Extractor	Extracts features from page layout (PAGE XML) and document image, for usage in quality estimation.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
73	PRImA Page Metadata Extractor	Extracts metadata from PAGE XML file, including region count per type, text content statistics, ...  <a href="http://www.primaresearch.org/tools/PAGEMetadataScanner">http://www.primaresearch.org/tools/PAGEMetadataScanner</a>
74	PRImA Page Text Extraction	Extracts the Unicode text content from a PAGE XML file  <a href="http://www.primaresearch.org/tools/ExtractorExporter">http://www.primaresearch.org/tools/ExtractorExporter</a>
75	PRImA Text Normalisation	Applies text replacement rules to the text content of page content files or text files. Converts ALTO XML, FR XML, or older versions of PAGE XML to the latest version of PAGE XML.  <a href="http://www.primaresearch.org/tools/PAGEConverterValidator">http://www.primaresearch.org/tools/PAGEConverterValidator</a>
76	PRImA PAGE to SVG Converter	Converts PAGE XML file with layout and text content to viewable SVG file.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
77	PRImA Page Validation	Validates PAGE XML with layout and text content against ground truthing rules and guide lines.  <a href="http://www.primaresearch.org/tools/PAGEConverterValidator">http://www.primaresearch.org/tools/PAGEConverterValidator</a>
78	PRImA PAGE Visualisation	Visualisation of page content  <a href="http://www.primaresearch.org/tools/Aletheia">http://www.primaresearch.org/tools/Aletheia</a>
79	PRImA Page Conversion	Converts ALTO XML, FR XML, or older versions of PAGE XML to the latest version of PAGE XML.  <a href="http://www.primaresearch.org/tools/PAGEConverterValidator">http://www.primaresearch.org/tools/PAGEConverterValidator</a>
80	PRImA Page Post- Processor	Post-processing of page layout objects (e.g. regions) using specific rules such as text replacement or re-OCRing.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
81	PRImA Region Extraction	Extracts region image snippets from a document page scan using regions defined in PAGE XML  <a href="http://www.primaresearch.org/tools/ExtractorExporter">http://www.primaresearch.org/tools/ExtractorExporter</a>
82	PRImA Table Classifier	Classifies/recognises the type of page content (e.g. a specific form or table) by comparing OCRred text with a range of known layouts (or rather the text content of the pages).  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
83	PRImA Table Exporter	Exports tabular page content to CSV  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
84	PRImA Text Line Extraction	Extracts text line image snippets from a document page scan using text lines defined in PAGE XML  <a href="http://www.primaresearch.org/tools/ExtractorExporter">http://www.primaresearch.org/tools/ExtractorExporter</a>
85	PRImA Text Line Segmenter	Segments text regions (blocks/zones) into text lines.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>



86	PRImA Typewritten OCR	OCR Prototype for recognising typewritten documents incorporating background knowledge about the specific features of this type of documents.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
87	PRImA Word Extraction	Extracts word image snippets from a document page scan using words defined in PAGE XML  <a href="http://www.primaresearch.org/tools/ExtractorExporter">http://www.primaresearch.org/tools/ExtractorExporter</a>
88	PRImA Word Segmenter	Segments text lines into words.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
89	PRImA PAGE to PDF	Converts PAGE XML (layout+text) to a PDF with hidden text layer and the corresponding document image.  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
90	PRImA OCR Evaluation Tool	Evaluation Tool for OCR by Pattern Recognition and Image Analysis (PRImA) research lab  <a href="http://www.primaresearch.org/tools/PerformanceEvaluation">http://www.primaresearch.org/tools/PerformanceEvaluation</a>
91	Sauvola Binarisation	Sauvola dynamic binarisation using local thresholds (part of PRImA Image Tool)  <a href="http://www.primaresearch.org">www.primaresearch.org</a>
92	Scan Tailor	Scan Tailor is an interactive post-processing tool for scanned pages. It performs operations such as page splitting, deskewing, adding/removing borders, and others.  <a href="http://scantailor.org/">http://scantailor.org/</a>
93	ShapeCatcher	Unicode character recognition! This is a tool to help you find Unicode characters. Finding a specific character whose name you do not know is cumbersome. On shapecatcher.com, all you need to know is the shape of the character! How do I use it? Draw your character as best you can in the "drawbox". You can do this by clicking and holding the left mouse button and moving around. Draw as many strokes as you need to, then click "Recognize" to start the recognition. If you want to clear the canvas and the results click on "Clear".  <a href="http://shapecatcher.com/">http://shapecatcher.com/</a>
94	SharpEye Musical Score Recognition	Music OCR: You can use SharpEye to scan and convert printed sheet music into a music notation file or a MIDI file which can then be imported into a music notation program or MIDI sequencer  <a href="http://www.visiv.co.uk/">http://www.visiv.co.uk/</a>
95	Stanford Parser	This package is a Java implementation of probabilistic natural language parsers, both highly optimized PCFG and lexicalized dependency parsers, and a lexicalized PCFG parser.  <a href="http://nlp.stanford.edu/software/lex-parser.shtml">http://nlp.stanford.edu/software/lex-parser.shtml</a>
96	Tesseract 3.03	Tesseract 3.03 with export to PRImA PAGE XML format.  <a href="http://www.primaresearch.org/tools/TesseractOCRtoPAGE">http://www.primaresearch.org/tools/TesseractOCRtoPAGE</a>
97	Text and Error Profiler	The Text and Error Profiler is software to analyse the OCR output from historical documents, using statistical modelling of document characteristics to improve OCR accuracy. It works by attuning itself to a particular document, rather than to common traits of printed documents from a certain era, resulting in a highly adaptive process. The tool uses its document-specific knowledge to allow the batch processing of erroneous words. During the IMPACT Project (2008-2011), a working group at the University of Munich developed software to analyse the OCR output from historical documents, using statistical modelling of document characteristics to improve OCR accuracy.  <a href="http://www.digitisation.eu/tools-resources/tools-for-text-digitisation/text-and-error-profiler/">http://www.digitisation.eu/tools-resources/tools-for-text-digitisation/text-and-error-profiler/</a>
98	tiffTool	TiffTool is a high-performance tool to clean scanned documents in preparation for onscreen display or for OCR. Features include skew correction, orientation correction, despeckle, page alignment, split pages and batch processing.  <a href="http://tiffTool.sourceforge.net/">http://tiffTool.sourceforge.net/</a>
99	Transcript	Transcript is a desktop-based manuscript transcription tool that supports word-processor style

		<p>formatting.</p> <p><a href="http://www.jacobboerema.nl/en/Freeware.htm">http://www.jacobboerema.nl/en/Freeware.htm</a></p>
100	TypeWright	<p>TypeWright is a tool for correcting the text-version of a document made up of page images.</p> <p><a href="http://www.18thconnect.org/typewright">http://www.18thconnect.org/typewright</a></p>
101	Unpaper	<p>Unpaper is a post-processing tool for scanned sheets of paper, especially for book pages that have been scanned from previously created photocopies. The main purpose is to make scanned book pages better readable on screen after conversion to PDF. Additionally, unpaper might be useful to enhance the quality of scanned pages before performing optical character recognition (OCR).</p> <p><a href="https://sourceforge.net/projects/unpaper/">https://sourceforge.net/projects/unpaper/</a></p>
102	Wavelet image denoising	<p>The wavelet denoise GIMP plugin is a tool to reduce noise in each channel of an image separately. The default colour space to do denoising is YCbCr which has the advantage that chroma noise can be reduced without affecting image details. Denoising in CIELAB (L*a*b*) or RGB is available as an option. The user interface allows colour mode and preview channel selection. The denoising threshold can be set for each colour channel independently.</p> <p><a href="http://registry.gimp.org/node/4235/">http://registry.gimp.org/node/4235/</a></p>
103	WordFreak	<p>WordFreak is a java-based linguistic annotation tool designed to support human, and automatic annotation of linguistic data as well as employ active-learning for human correction of automatically annotated data. Java based.</p> <p><a href="http://wordfreak.sourceforge.net/">http://wordfreak.sourceforge.net/</a></p>
104	ZBar	<p>ZBar is an open source software suite for reading bar codes from various sources, such as video streams, image files and raw intensity sensors. It supports many popular symbologies (types of bar codes) including EAN-13/UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2 of 5 and QR Code.</p> <p><a href="http://zbar.sourceforge.net/">http://zbar.sourceforge.net/</a></p>

## Appendix D – Publication

Below is a copy of a peer-reviewed publication which was selected for oral presentation at the Document Analysis Systems workshop DAS2018 in Vienna, Austria.

# Ontology and Framework for Semantic Labelling of Document Data and Software Methods

Christian Clausner and Apostolos Antonacopoulos  
Pattern Recognition and Image Analysis Research Lab  
University of Salford  
United Kingdom  
www.primaresearch.org

**Abstract**— We present a metadata labelling framework for datasets, software tools, and workflows. An ontology for document image analysis was developed with deep support for historical data. An accompanying open source software framework was implemented to enable ontology editing, data and method annotation, workflow composition, and semantic search. A wide range of examples is used to illustrate real-world application.

**Keywords**- *Semantics; ontology; historical documents; document retrieval; scientific workflows; metadata*

## I. INTRODUCTION

With the rapidly expanding volume of data and increasing complexity of software it becomes increasingly important to establish sophisticated annotation and retrieval systems. Several metadata formats (e.g. METS [1] and Dublin Core [2]) with varying amount of detail are used to add information about authorship, publication date, technical characteristics etc. to data items such as documents and the corresponding document images.

We present a complementary framework to annotate data, but also software methods targeting specific data, using semantic labels describing the nature of the items in detail. This is part of an ongoing research project investigating scientific workflows including workflow composition systems and repositories.

Scientific workflows are used to automate software processes. To that effect, workflow systems typically offer the following features: visualisation, fault tolerance, distribution, data provenance, and repeatability. Workflows are defined by their activities (actors) and the way they are connected (data flow). Activities therein have data input and output ports.

To allow for more automation and assistive features in workflow composition and retrieval, a semantic labelling approach was developed, including a complete framework for: ontology creation and editing, workflow composition and labelling, semantic matching algorithms, and data and workflow repositories.

To the authors' knowledge, the proposed framework is the first implementation of a flexible and yet powerful semantic labelling approach (not only for documents). Other workflow systems with semantic features ([14][20]) use very strict semantic models (a workflow has to be modelled as a whole using semantic "language") and default reasoners currently limited to basic queries.

Workflow data items and online datasets have in common that a detailed semantic description helps with search and retrieval. We therefore propose that the

developed ontology and labelling approach can be used to extend the existing metadata of document image datasets. The ontology includes a wide variety of concepts, complementing existing schemes which usually represent an archiving / library point of view or a very technical point of view (e.g. image attributes).

The proposed ontology and its creation are described in the next section, followed by a description of the software framework in Section III. After a discussion in Section IV the paper concludes with Section V.

## II. ONTOLOGY

The ontology was developed for document image analysis and recognition, but with extensibility to other domains in mind. METHONTOLOGY [3] was used as design strategy. As the initial conceptualisation, domain-related terms were collected from various sources (text books, IJDAR papers, ACM classification scheme, project reports etc.) and put into a "term cloud". The mostly randomly arranged terms were then moved and grouped iteratively until high- and low-level concepts emerged.

To include concepts for historical material, a collection of keywords for tagging of document images was incorporated as well. They originate from the IMPACT project [4] and were refined during the European Newspapers project [5]. An early description can be found in a keynote by Antonacopoulos [6]. The keywords were used to tag existing datasets [7][8] and range from document characteristics to flaws/conditions introduced through ageing, wear, or during digitisation. The original categorisation was refined to fit in with other concepts in the ontology.

The keywords are in line with related work for defects in printed documents [9][10] and photographs [11]. It is worth noting that certain aspects are modelled in more domain-specific detail in those works (e.g. grouping into physical, chemical, and biological cause of degradation or conditions specific to developed photographs), which can be used as basis for future extensions of the proposed ontology, where required.

The ontology consists of *label type* hierarchies, each targeting one aspect of a data item or an activity. A label type is thereby a concept within a class hierarchy. Labels are an instance of a label type, when attached to a data item (as metadata). In other words, the ontology is a collection of taxonomies (or partonomies), with several (more general) root label types and attached trees of label types which represent more specialised terms.

Both *activities* and *data objects* can be labelled. Here, an activity is any kind of process that transforms or

generates data. A data object can be the input for or output from an activity, but also a standalone item (such as a document image) or a collection (such as a document image dataset). Partial labelling (i.e. for sub-elements of a data object) can be realised as well, but is not currently part of the implementation.

The top-level label types for activities are: domain, processing level, automation, data creation, adaptability, platform, and licence. Activities are also characterised by their input and output data, which can be labelled as well. When searching for an activity (e.g. a software tool), it can be taken into account whether it processes images of a certain kind, whether it is fully automated, whether it runs on a given operating system, and whether it produces a desired output (for instance).

The top-level types for data objects are: source, age, physical production method, acquisition method / replication steps, precision, content type, content encoding, source / target content, data granularity, data condition, data attributes, and topic. TABLE I. shows an abbreviated list of data-related label types.

The complete ontology also contains definitions and examples for terms. Currently it consists of about 350 label types. An ontology version and the option of migration rules allows for future extensions without breaking backwards compatibility. The ontology can be serialised using the OWL (Web Ontology Language) standard [12] or a simplified XML format.

The next section describes how labels are used to search for objects or aid the creation of workflows.

### III. FRAMEWORK

A range of algorithms and supporting software tools (Java-based) was developed to test and demonstrate the proposed labelling approach. In this section, the different components of that framework are described.

The *Ontology Editor* is used to view and edit the label types using a tree-based interface. In addition to types, label slots are used to define how many labels of one category can be assigned to a data object or activity (see Figure 1. ). Export to OWL format allows the ontology to be viewed (and edited) also in other general-purpose tools such as Protégé [13] (although editing is simpler in the dedicated tool). It should be noted that the editor can be used to extend the ontology proposed in Section II or to create an entirely new one. Versioning is supported via a single integer number.

The *Workflow Editor* allows the composition of scientific workflows. The approach is loosely based on the ASKALON workflow system [14]. Different types of activities are combined to create control flow. Data flow is achieved through connecting data input and output ports of activities. The ports and also the activities can be annotated using the semantic labels.

Workflow, activity, and data repository tools (within the *Repository Hub*) are used to aggregate collections of the respective objects. They include semantic and keyword-based search functionality. The search interface is closely related to the label type hierarchies and works by gradually refining results using tick boxes (in a similar fashion to the faceted-search of online catalogues or shops).

A matching algorithm compares a given set of labels (e.g. for a data port) to a collection of label sets (e.g. from a data collection). It calculates a match score for each pair of label sets, allowing for partial matches (i.e. when a label on one side is of a higher level but similar category that a label on the other side). The more the “search labels” match the labels of a searchable object the higher is the match percentage (each non-match reducing the match score heuristically). This allows for flexible searches where the results are sorted by match score (high to low) and not restricting the search to 100% matching objects. Match details show which aspects (labels) match well and which do not. In addition to labels, also data types can be included in the search. Data type matching is strict and requires exact matches. Apart from a few primitive types (string, integer, float), data types are user-defined.

The source code and UML diagrams can be found at the authors’ GitHub profile [22].

In general, labelling and matching are used to add assistive features to the workflow framework (search, composition, validation etc.).

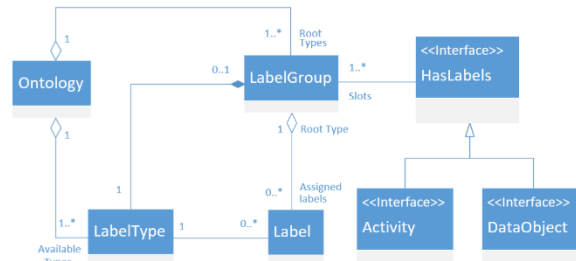


Figure 1. Ontology and labels (UML diagram). A label group represents sub-tree of the ontology for a specific root label type (i.e. a category). An object that can be labelled has slots for labels from different categories. Activities (i.e. software methods) and data objects can be labelled (have the HasLabels interface). The ontology represents the label type hierarchy but has no actual Label instances. Label instances (each having a specific label type) are attached to objects with the HasLabels interface.

### IV. DISCUSSION

This section describes general thoughts, application scenarios and labelling examples.

Although the framework allows the creation of new ontologies, better interoperability can be achieved by using the proposed ontology and extending it where necessary. Such extension activities would require central access and a consortium or a community which manages changes (including a full life cycle with versioning). Problems arising from the size of the ontology (high quantity of label types) can be solved on the user interface layer by only using a subset of labels depending on the domain of the application and by providing convenience tools such as a label quick search (e.g. by keyword). Backwards compatibility of already labelled datasets can be achieved by using migration rules that are part of the ontology itself (already implemented within the software framework).

#### A. Applications

Existing document image datasets could be annotated using the proposed label-based approach. Both individual items (images, ground truth files) and complete datasets

can be labelled. A dataset could also have a combination of top-level labels to allow searching across different datasets, for example. Individual items could be labelled in more detail. A common use case is search and retrieval of training data for document image analysis methods. Often, data of a very specific nature is required, which, at the moment, can only be searched for by keywords (e.g. using Internet search engines).

The labelling scheme can be combined with existing description mechanisms such as METS [1], the International Image Interoperability Framework (IIIF) [15], or the PAGE format [16]. For this purpose, labels can be stored in a serialised form:

<ID Level 1>.<ID Level 2>.<ID Level 3>...

The matching and search algorithms can be applied independently from the overall software framework. Ontology versioning should be kept in mind and the version number should be stored together with the labels.

In addition to data repositories, semantic labelling can also enhance *software* and *workflow repositories* such as those of the IMPACT Centre of Competence for Digitisation in the EU [17] or the myExperiment platform [18], which typically only allow a search by keywords or by category. Semantic labels could make a significant difference in the ease of workflow composition by describing input and output data of tools as well as properties of the methods themselves.

The primary use of the proposed ontology and framework is in workflow composition and retrieval. While the developed framework is fully functional with respect to workflow composition, it does not contain a workflow execution (enactment) component. Instead of implementing such a component, the labelling approach could be ported to existing workflow systems such as Taverna [19] or Kepler [20].

### B. Examples

This subsection provides selected examples for image conditions taken from the Europeana Newspapers Dataset [7] and the Census 1961 dataset [8]. The properties and conditions were chosen because they can have influence on recognition performance or require specialised methods.

Following selection of image characteristics and conditions from different categories are illustrated below:

- Data properties – Intended / production-related features (Figure 2.)
- Data condition – Unintended properties / flaws / issues
  - Production-related – Limitations of production method or imperfections (Figure 3.)
  - Wear/use – Problems due to (heavy) use (Figure 4.)
  - Ageing – Storage conditions or exposure (Figure 5.)
  - Acquisition- or conversion-related – Problems introduced by copying or digitisation (Figure 6.)

A given image from the Europeana set can be labelled as follows (label type hierarchy: top level – ... – lower level):

- Original source – produced data – physical medium – paper document – newspaper
- Age – historical

- Physical production method – printed – typeset
- Content type – data
- Content encoding – raster image – colour
- Content of interest – visual – text
- ... – visual – graphical
- Topic – Economy – financial / business
- Data granularity – physical – page
- Data properties – language – mixed languages
- ... – document-related – visual – text – font – typeface class – blackletter
- ... – text – font – multi-font – mixed typefaces
- ... – text – font – multi-font – mixed font sizes
- ... – columns – multiple
- Data condition – noisy – speckles
- ... – production-related – doc. characteristics – halftoning
- ... – flaws – touching characters – horizontally
- ... – flaws – broken characters
- ... – wear / use – medium damage – folds
- ... – wear / use – additions – stamps
- ... – ageing – warping
- ... – ageing – discolouration
- Acquisition- / conversion-related – geometric – perspective distortions
- ... – background-related – included parts / objects – paper clips
- ... – method flaws – imaging-related – show-through

#### Data Attributes / Properties



Figure 2. Selected image / document conditions and properties for category “Data attributes / properties”. Percentages indicate how widespread the properties are in the Europeana Newspapers set.

Finally, it should be noted that the labelling approach is not only suitable for images but also for related data such as page ground truth. A ground truth file can be labelled with:

- Original source – produced data – physical medium – paper document – newspaper
- Content type – data
- Content type – metadata

- Content encoding – structured
- Content of interest – visual – text
- Content of interest – visual – graphical
- Topic – Economy – financial / business
- Data granularity – physical – page
- Data granularity – physical – region / zone
- Data properties – language – mixed languages

Data Condition		
Production-related		
Pasted clippings 	Textured paper (1%) 	Narrow margin (0.5%) 
Low text contrast (2%) 	Halftoning (10%) 	Uneven ink (25%) 
Bleed-through (4%) 	Ink from facing page (0.2%) 	Broken characters (44%) 
Faint characters (15%) 	Blurred chars (16%) 	Smearred ink (3%) 
Filled-in chars (40%) 	Sort shoulder artef. (15%) 	Touching chars (44%) 

Figure 3. Selected image / document conditions and properties for category “Data condition – production-related”. Percentages as in Europeana Newspapers set.

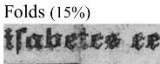

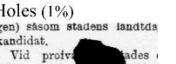
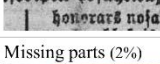
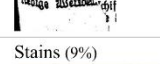
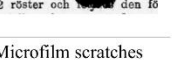




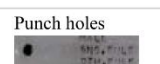
Wear / use		
Folds (15%) 	Tears (7%) 	Holes (1%) 
Missing parts (2%) 	Stains (9%) 	Microfilm scratches (5%) 
Paper repairs (4%) 	Punch holes 	Annotations (5%) 
Handwritten correction 	Stamps (8%) 	

Figure 4. Selected image / document conditions and properties for category “Data condition – wear / use”. Percentages as in Europeana Newspapers set.

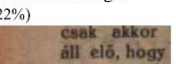
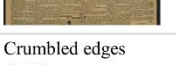

Ageing		
Warped paper (11%) 	General paper discolouration (10%) 	Discoloured edges (22%) 
Mould 	Crumbled edges (0.5%) 	Fading ink (10%) 

Figure 5. Selected image / document conditions and properties for category “Data condition - ageing”. Percentages as in Europeana Newspapers set.


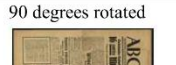




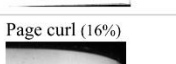
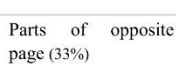
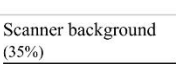
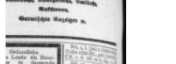


Digitisation / copying		
Skew (9%) 	90 degrees rotated 	Upside down 
Page curl (16%) 	Parts of opposite page (33%) 	Scanner background (35%) 
Uneven illumination (9%) 	Low scan contrast (1%) 	Paper clips (2%) 
Salt-and-pepper noise (18%) 	Missing information after binarisation (7%) 	Binarisation noise (9%) 

Figure 6. Selected image / document conditions for category “Data condition – digitisation / copying”. Percentages as in Europeana Newspapers set.

## V. CONCLUDING REMARKS AND FUTURE WORK

A semantic labelling framework for data and software methods was presented. The labels describe the nature of objects in different aspects and can be used in conjunction with existing metadata formats.

The ontology, currently targeting document analysis and recognition, could be extended to other domains and

complemented with further languages and scripts (e.g. using IDs from ISO standards), for instance. The intention is to offer a generic model which applies to many use cases and is open for extensions and specialisations. Further semantic concepts such as relations and literals can be added to the model if necessary.

As a demonstration, an existing public dataset (the Europeana Newspapers set [7], for instance) might be labelled using the proposed scheme and an online interface can be developed to access the information.

The complete version of the presented ontology and the software framework are available on GitHub [22].

TABLE I. DATA OBJECT LABELS (235 OUT OF 350 TOTAL LABELS FOR DATA AND SOFTWARE METHODS). HIGH-LEVEL TYPES ON THE LEFT, LOWER-LEVEL TYPES ON THE RIGHT.

Lev.1	Lev. 2	Lev. 3	Lev. 4	Lev. 5	Lev. 6	Lev. 7
<b>Original Source</b>						
	Produced data					
		Physical source	medium			
			Paper document			
				Book		
				Newspaper		
	Captured data					
		Real / natural scenes				
<b>Age</b>						
	Historical					
		Medieval				
	Contemporary					
	Ancient					
<b>Physical Production Method</b>						
	Manual					
	Machine					
		Printed				
			Typeset			
			Computer printout			
				Typewritten		
<b>Acquisition / Replication Method</b>						
	Analogue / physical to digital					
		Scanning				
		Camera				
	Copied					
		Photocopy				
		Microfilm / microfiche				
	Synthesis					
<b>Precision</b>						
	Ground Truth / gold standard					
	Measured					
	Estimated					
	Random					
	Fuzzy					
<b>Content Type</b>						
	Data					
	Metadata					
		Quality				
			Performance Information			
		Features				
		Structure				
			Table of contents			
		Annotations				
		Authorship				
		Spatial				
			Location			
	Settings					
	Model					
	Lexicon / index					
	Corpus / database					
<b>Content Encoding</b>						
	Textual					
		Annotated				
	Structured					
	Raster image					
		Colour Image				
		Bitonal				
	Mathematical / geometrical					
		Vector-based				
			Stroke-based			
	Polygonal					
<b>Content of Interest</b>						
	Visual content					
		Text				
		Graphical				
			Separators			
			Barcode / QR Code			
		Image				

		Photograph			
		Person(s)			
		Drawing			
	Mixed / composite content				
		Tables / forms			
		Charts			
		Maps / plans			
		Mathematical expression			
<b>Data Granularity</b>					
	Physical / visual granularity				
		Document-related			
		Double-page			
		Page			
		Region / Zone			
		Text line			
		...			
		Natural language-related			
		Sentence			
		Token / chunk			
		Syllable			
	Logical granularity				
		Document-related			
		Document			
		Chapter			
		Section			
		Article			
		Paragraph			
<b>Data Condition</b>					
	Noise				
		Speckles			
			Salt-and-pepper noise		
		Clutter			
			Thresholding-related noise		
	Production-related				
		Document characteristics			
		Pasted clippings			
		Textured paper			
		Uneven character spacing			
		Narrow border			
		Low paper-to-content contrast			
		Half-toning			
		Dithering			
		Document faults			
		Bleed-through			
		Ink from facing page			
		Smeared ink			
		Touching characters			
			Horizontally		
			Vertically		
		Uneven ink distribution			
		Filled-in characters			
		Sort shoulder artefacts			
		Broken characters			
		Faint characters			
		Blurred characters			
		Non-straight text lines			
	Wear / use				
		Medium damage			
		Folds			
		Tears			
		Holes			
			Punch holes		
			Unintended holes		
		Missing parts			
		Stains			
		Scratches			
		Staples			
		Additions			
		Visible repairs			
			Paper repairs		
			Clear tape		
		Informative			
			Annotations		
			Stamps		
		Corrections			
			Manual corrections		
	Ageing				
		Warping			
		Discolouration			
			Global		
			Edges		
		Disintegration			
			Uneven edges		
		Mould			
		Fading content			
	Acquisition / conversion-related issues				
		Geometric issues			
			Skew		
				Global	
				Non-uniform	
				90-degree rotation	



			Upside down	
			Perspective distortions	
			Page curl	
	Content / background-related			
			Incomplete capture	
			Tight / narrow margins	
			Included other objects	
			Part of pre- or succeeding object	
			Medium structure (book cover...)	
			Paper clips	
			Fingers	
			Insects	
			Background (e.g. scan bed)	
	Method flaws			
			Imaging-related	
			Show through	
			Uneven illumination	
			Shadows	
			Out-of-focus	
			Low contrast	
			Missing / changed content	
			Due to thresholding	
<b>Data Attributes / Properties</b>				
	Language			
		Natural language		
			English	
		Mixed languages		
	Document-related			
		Visual properties		
			Text-related	
				Script
				Braille
				Latin
				Font-related
				Cursive
				Monospace
				Hand / Typeface class
				Blackletter
				Antiqua
				Medieval manuscript
				Decorated text
				Flourishes
				Multiple colours
				Reverse video
				Multi-font
				Mixed typefaces
				Mixed font sizes
				Drop caps
				Columns
				One column
				Two columns
				Multiple columns
				Rotated content
				Complex background
				Watermarks
				Impressions / embossing
				Illustrations
				Multi-coloured
				Decorations
				Frames / borders
				Line drawing / line-art
				CAPTCHAs
	Structural			
				Running titles
				Footnotes
				Bibliographic reference
<b>Topic</b>				
	Economy			
		Financial / business		
			Bank checks	
			Invoices	
	Social science / environmental			
		Maps		
			Topographical maps	
			Road maps	
		Traffic and automotive		
			Number plates	
			Traffic signs	
	Science and Engineering			
		Architectural		
			Floor plans	
			Architectural drawings	
		Medical		
		Engineering drawings		
		Patents		
	Media / entertainment			
		Advertisements		
	Computing			

## REFERENCES

- [1] J. P. McDonough, "METS: standardized encoding for digital library objects" *Int. Journal on Digital Libraries* 6, 2, pp. 148–158, Apr. 2006, DOI: <https://doi.org/10.1007/s00799-005-0132-1>
- [2] Dublin Core Metadata Initiative, <http://dublincore.org/>, accessed 09/08/2017
- [3] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: From Ontological Art Towards Ontological Engineering", AAAI-97 Spring Symposium Series, 1997, American Association for Art. Int.: Stanford University, EEUU.
- [4] IMPACT Project 2011 Publishable summary. <http://www.impact-project.eu/documents>, 2012.
- [5] Europeana Newspapers Project., Staatsbibliothek zu Berlin, <http://www.europeana-newspapers.eu>, [cited 2016 01/06]
- [6] A. Antonacopoulos, "Large-Scale Digitisation and Recognition of Opportunities for Image Processing and Analysis Historical Documents: Challenges and Analysis", Keynote presentation given at the Swedish Symposium on Image Analysis 2010 (SSBA2010), Uppsala, Sweden, March 11-12, 2010.
- [7] C. Clausner, C. Papadopoulos, S. Pletschacher, A. Antonacopoulos, "The ENP Image and Ground Truth Dataset of Historical Newspapers", *Proc. 13th Int. Conf. on Document Analysis and Recognition (ICDAR2015)*, Nancy, France, 08/2015, pp. 931-935.
- [8] C. Clausner, J. Hayes, A. Antonacopoulos, S. Pletschacher, "Unearthing the Recent Past: Digitising and Understanding Statistical Information from Census Tables", *Proc. Second International Conference on Digital Access to Textual Cultural Heritage (DATECH)*, Goettingen, Germany, 01-02 June 2017.
- [9] R. D. Lins, "A Taxonomy for Noise in Images of Paper Documents - The Physical Noises", *Proceedings of the 6th Int. Conf. on Image Analysis and Recognition*, Halifax, Canada, July 6-8, 2009.
- [10] H. S. Baird, "The State of the Art of Document Image Degradation Modelling", Book chapter in *Digital Document Processing: Major Directions and Recent Advances*. Springer, Lon., 2007, p. 261-279.
- [11] E. Ardizzone, A. De Polo, H. Dindo, G. Mazzola, C. Nanni, "A Dual Taxonomy for Defects in Digitized Historical Photos", *10th International Conference on Document Analysis and Recognition, ICDAR 2009*, Barcelona, Spain, 26-29 July 2009.
- [12] W3C, *OWL 2 Web Ontology Language Primer (Sec. Ed.)*. 2012
- [13] Protégé - A free, open-source ontology editor and framework for building intelligent systems. [cited 2015 24/03].
- [14] J. Qin, T. Fahringer, "Scientific workflows : programming, optimization, and synthesis with ASKALON and AWDL", 2012, Berlin, New York: Springer. xxi, 222 pages.
- [15] International Image Interoperability Framework – IIIF, <http://iiif.io>
- [16] S. Pletschacher and A. Antonacopoulos, "The PAGE (Page Analysis and Ground-Truth Elements) Format Framework", *Proc. ICPR2008*, Istanbul, Turkey, 2010, pp. 257-260.
- [17] IMPACT Centre of Competence for Digitisation, <https://www.digitisation.eu>
- [18] D. Roure, C. Goble, R. Stevens, "The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows", *Future Generation Computer Systems*, 2008. 25: p. 7.
- [19] T. Oinn et al., "Taverna: a tool for the composition and enactment of bioinformatics workflows", *Bioinformatics*, 2004. 20(17): p. 3045-3054.
- [20] B. Ludascher et al., "Scientific workflow management and the Kepler system: Research Articles", *Concurr. Comput. : Pract. Exper.*, 2006. 18(10): p. 1039-1065.
- [21] Y. Gil et al., "Expressive reusable workflow templates", *e-Science*, 2009, e-Science'09. Fifth IEEE International Conference on. 2009. IEEE.
- [22] Semantic labelling software framework GitHub repository, <https://github.com/PRImA-Research-Lab/semantic-labelling>