

Understanding abstract geo-information workflows and  
converting them to executable workflows using  
Semantic Web technologies

**MSc Thesis**

**Sam Ubels**

**March 2018**

Professor: Prof. Dr. MJ. Kraak,  
Department of Geoinformation Processing  
Faculty of Geoinformation Science and Earth Observation  
University of Twente

Supervisor: Dr. Ir. Rob Lemmens  
Department of Geoinformation Processing  
Faculty of Geoinformation Science and Earth Observation  
University of Twente



## Contents

Acknowledgements.....	3
List of figures.....	4
List of listings.....	5
List of tables.....	5
List of abbreviations.....	6
Abstract.....	8
1. Introduction.....	9
1.1 Solving spatial problems made easier.....	9
1.2 Problem statement.....	13
1.3 Research objectives.....	14
1.4 Use Case.....	17
1.5 Research Limitations.....	18
1.6 Reading guide.....	18
2 Semantic Web.....	19
2.1 Linked Data.....	19
2.2 Resource Description Framework.....	19
2.3 Ontologies.....	23
2.4 SPARQL.....	24
2.5 Semantic Web Tools.....	25
3 Workflows.....	28
3.1 Scientific Workflows.....	29
3.2 Sharing Workflows.....	30
3.3 Levels of abstraction.....	31
3.4 Semantic Web to the rescue.....	35
4 Proof of concept.....	39
4.1 The approach.....	39
4.2 The ontology.....	43
4.3 The Application.....	45
5 Analysis of results.....	52
5.1 Limitations.....	52
5.2 The abstract workflow.....	52
5.3 The ontology.....	53
5.4 The Application.....	54

5.5 The output.....	56
6 Conclusions and recommendations.....	60
6.1 Conclusions .....	60
6.2 Recommendations .....	66
Bibliography .....	67
Appendices.....	71
Appendix 1: Geo-operation ontology .....	71
Appendix 2: Abstract workflow .....	81
Appendix 3: Executable Workflow ILWIS.....	84
Appendix 4: Executable workflow ArcMap.....	86
Appendix 5: User Manual for proof of concept application .....	88

## Acknowledgements

Through here, I would like to extend my sincere gratitude to my supervisor dr.ir. R.L.G. Lemmens, who stood up to my personal brand of time management. I would like to thank him for his ongoing guidance during this thesis project, and help in moments where I wasn't sure how to continue. I would also like to thank my girlfriend, Godelief, who kept supporting me for the entire duration of my writing process. Finally, I would like to thank my father, Jan Bert, who was the selected victim for all the necessary proof reading, brainstorm sessions and functioned as a sounding board whenever it was needed.

Thanks to all, I couldn't have done it without you!

## List of figures

### Chapter 1

<i>Figure 1.1: Spatial problem-solving application</i> .....	11
--	----

### Chapter 2

<i>Figure 2.1: Example of RDF structure</i> .....	20
---	----

<i>Figure 2.2: RDF example using URIs</i> .....	21
---	----

### Chapter 3

<i>Figure 3.1: Overview model of provenance information stored in the PROV standard</i> .....	29
---	----

<i>Figure 3.2: Original Use Case Workflow</i> .....	32
---	----

<i>Figure 3.3: Schematic representation of abstraction levels</i> .....	33
---	----

<i>Figure 3.4: Connection between workflow ontologies</i> .....	35
---	----

<i>Figure 3.5: Spatial problem-solving application</i> .....	36
--	----

### Chapter 4

<i>Figure 4.1: Schematic overview of original Use Case Workflow</i> .....	40
---	----

<i>Figure 4.2: Abstract version of Use Case Workflow</i> .....	41
--	----

<i>Figure 4.3: Schematic overview of the GeoOperation Ontology</i> .....	43
--	----

<i>Figure 4.4: Functional model of the proof of concept application</i> .....	47
---	----

<i>Figure 4.5: Workflow upload and software selection screen</i> .....	48
--	----

<i>Figure 4.6: Screenshot of the proof of concept application providing the user with overview of workflow operations and documentation</i> .....	49
---	----

<i>Figure 4.7: Screenshot of the proof of concept application showing the user input fields</i> .....	50
---	----

### Chapter 5

<i>Figure 5.1: Geo-Operation ontology</i> .....	53
---	----

<i>Figure 5.2: Output ILWIS Workflow</i> .....	57
--	----

<i>Figure 5.3: Output ArcMap Workflow</i> .....	58
---	----

## List of listings

### Chapter 2

<i>Listing 2.1: Example of RDF/XML serialisation</i> .....	22
<i>Listing 2.2: Example of turtle serialisation</i> .....	22
<i>Listing 2.3: Example of JSON-LD serialisation</i> .....	23
<i>Listing 2.4: Example of a SPARQL select query</i> .....	25

### Chapter 4

<i>Listing 4.1: RDF instantiation of Polygon to Raster operation</i> .....	44
<i>Listing 4.2: Original output ILWIS model builder vs edited output</i> .....	45
<i>Listing 4.3: Workflow logic as created by ILWIS model builder</i> .....	46
<i>Listing 4.4: Dynamic query builder</i> .....	48
<i>Listing 4.5: Build function to dynamically create html page</i> .....	49
<i>Listing 4.6: Query function to acquire operation data</i> .....	49
<i>Listing 4.7: Function creating Python variables from user input</i> .....	51
<i>Listing 4.8: Code snippet to set up executable ArcMap workflow</i> .....	51

### Chapter 5

<i>Listing 5.1: Code snippet to set up executable ILWIS workflow</i> .....	55
<i>Listing 5.2: Code snippet to set up executable ArcMap workflow</i> .....	55

## List of tables

### Chapter 4

<i>Table 4.1: Overview of parameters and their description</i> .....	50
--	----

## List of abbreviations

AI – Artificial Intelligence

API – Application Programming Interface

DBMS – Database Management System

DCAT – Data Catalog Vocabulary

GIS – Geographic Information System

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

ILWIS – Integrated Land and Water Information System

ITC – Faculty of Geo-Information Science and Earth Observation

JSON – JavaScript Object Notation

JSON-LD – JavaScript Object Notation Linked Data

MaMaSe – Mau Mara Serengeti

OGC – Open Geospatial Consortium

OPM – Open Provenance Model

OPMW – Open Provenance Model Workflows

OWL – Web Ontology Language

PROV – Provenance Model

PROV-O – Provenance Model Ontology

PROV-Wf – Provenance Model Workflow extension

PSI – Public Sector Information

RDF – Resource Description Framework

RDFS – Resource Description Framework Schema

SKOS – Simple Knowledge Organisation System

SPARQL – Simple Protocol and RDF Query Language

SQL – Structured Query Language

SWfMS – Scientific Workflows Management System

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

WfMS – Workflow Management System

WFMC – Workflow Management Coalition

WKT – Well Known Text

XML – Extensible Markup Language



## Abstract

The solving of spatial problems is something people deal with every day. Ranging from very simple questions, to very complex ones. Where expert users need to overcome many hurdles in solving spatial problems, this task becomes sheer impossible for non-expert users. This is mainly caused by the proprietary nature of much of the data and analysis tools available, lacking interoperability and due to a very broad range of data formats. Sharing solutions to spatial problems in the form of reusable workflows is a step in the right direction, but here a similar issue applies. Software specific workflow features, proprietary formats and a lack of metadata and workflow descriptions, seriously hinder the reuse. A proposed solution for making workflows more reusable is the use of abstract workflows, however in this case, again standardisation is required. A solution to these standardisation issues can be found in the Semantic Web.

This main objective of this thesis is the development of a method to semantically enrich an abstract workflow and convert it to an executable workflow using Semantic Web technologies. As a proof of concept, a workflow of the MaMaSe project has been used, calculating the total rainfall in the project area, for the entire rain season.

This proof of concept required several steps and elements. First, the creation of an abstract workflow, used as an input to the application. An ontology has been developed which provides a standardised structure to describe geo-operations. Based on this ontology, RDF descriptions of the geo-operations in the workflow have been created. These descriptions, including documentation, input and output parameters and additional information, are served to the user in a software application helping them understand the workflow and guiding them to provide the necessary input values required to execute the geo-operations used in the workflow. Based on user input, this workflow is converted to an executable python format using the python libraries of the execution engines of ArcMap and ILWIS.

The work has proven that standardisation using Semantic Web technologies provides a platform and software independent way to understand abstract workflows and guide users in the conversion to an executable workflow. The output of the workflows, executed in both execution engines, provide similar results, showing that the solution is indeed software independent.

## 1. Introduction

Every day people encounter spatial problems. These problems can range from a very simple specific spatial problem like ‘where is the nearest supermarket’, to more complex questions like ‘what is the best crop to grow on the piece of land I just bought’. To solve these questions a lot of spatial data is needed, and luckily this data is available, and even the methods already exist to come to an answer to some of these questions. However, in many cases people are not able to come to a clear and satisfactory answer to the more complex questions. This means that these people are somehow unable to find the method, tools and/or data that can be used to obtain an answer to their question. Hu & Li (2017) state that even though the amount of geospatial data, services and maps keep growing and are available in large numbers, this does not necessarily ease the use of these geospatial resources. This is caused by their often distributed and proprietary nature, and calls for a more open access to these resources. They are not alone in this, more and more companies, organisations and governments are seeing the need for an increase in open data and the benefits of sharing their data. Especially the sharing of government data is a growing phenomenon. Public bodies are often large creators and collectors of data and releasing this data may play a large part in innovation and creating new applications (Janssen, 2011). It is clear that limited availability of open data and tools is a stumbling block on the way to solving spatial problems. However, even when data is openly available this does not mean that easy reuse is guaranteed. Often data is not served in machine readable formats and adhering to metadata standards (Veeckman et al., 2017). This severely limits the interoperability of open data. Ma (2015) backs up this statement saying that merely retrieving a copy of a dataset does not mean the user understands its meaning, or knows how to use it. He calls this the ‘challenge of data interoperability’. Meaning that data should be ‘discoverable, accessible, decodable, understandable and useable’.

### 1.1 Solving spatial problems made easier

In order to take a step forward in allowing people to more easily solve their spatial problems, openness and standardisation are important factors. Both aspects of the geospatial data domain are discussed in more detail in this section.

#### Openness

The availability of open data as mentioned above is a crucial part of in allowing people to solve their spatial problems. However, the need for more open data is of course not a concern only observed when looking at solving day to day spatial problems. Therefore, in 2003, the European Union adopted the Public Service Information (PSI) directive, intended to stimulate member states to open up their government data to the public (European Parliament and Council, 2003). Even though efforts like this are a step in the right direction, expansion of open data is needed. Outside the borders of the EU as well as outside of the public domain. However, open data is not where it ends. To use and analyse the acquired data access to analysis tools is required as well. Especially, if non-expert users are to solve their spatial problems then free-software or web-services need to be available to perform spatial analysis. Luckily, when it comes to open source GIS software, less is needed. There already are a lot of open source GIS-clients and the number of online GIS services like Web Processing Services (WPS) are increasing as well (Steiniger & Hunter, 2013). Furthermore, easy ways to share the methods that are created to solve a spatial problem, so called workflows, should be made available. Workflows describe the different steps required to come to a specific result. To manage these workflows so called Workflow Management Systems (WfMS) were created, however most of these systems have a variety of particular features, creating a lack of interoperability, losing essential features of the workflow

when used in a different environment (Salado-Cid & Romero, 2016). So with workflows, as with data, the ability to share is not enough, if interoperability issues are not solved.

### Standardisation

When looking at standardisation to increase interoperability, in the context of solving spatial problems, three aspects are important; data, processing and workflows. When it comes to workflows first of all, standardisation should allow for easier sharing and better findability. Currently, the methods for sharing and publication of workflows is lacking. In many cases the different steps of a workflow are merely described in the research publications. However, research has shown that this rarely allows other scientists to reproduce the workflow in a way that is useful (Fang & Casadevall, 2011). Furthermore the workflows are only executable in a very specific combination of software, libraries and code, this means that even if the workflow is reusable it often does not allow the reuse in different circumstances or in different execution environments (Garijo & Gil, 2012). This means that to reuse workflows produced by others it is not only necessary to improve the findability but also finding a way to make the workflows software independent. A way to do this is using abstract workflows as proposed by (Garijo & Gil, 2012). They see abstract workflows as a 'conceptual and execution-independent view of the data analysis method'. As the application is expected to work with workflows that can be executed in different execution environments the use of an abstract workflow that describes the solving of a spatial problem in an execution-independent manner seems desirable. How to approach this will be discussed in more detail in chapter 3.

When it comes to standardisation of the operations comprising the workflows, the aim should be a standardised description of all geo-operations and a standardised set of parameters that are mandatory for each execution environment. Developers of GIS software can then add optional parameters to the operations as they see fit. Standardisation of the mandatory parameters allows the creation of workflows that will be executed in a similar way in all different execution environments. The call for this type of standardisation, allowing comparability of GIS analysis functionality and their meaning has been around already for a long time, however the Open Geospatial Consortium (OGC) just recently started to address this, for example with the OGC Web Processing Service 2.0 standard published in 2015 (Brauner, 2015).

The standardisation of the data or more specifically the metadata is probably the biggest challenge as there is a never-ending list of data producers who in many cases use a unique way of describing their data. Similar to the PSI directive which promotes the open sharing of government data there is the INSPIRE directive. The goal of INSPIRE is to create a European Spatial Data Infrastructure (SDI) which ensures that public bodies in EU member states all store, disclose and maintain their data in a similar way to make sure it is possible to combine the data without the need for extensive conversion (Vandenbroucke et al, 2008). Its aim being 'that it is easy to discover available spatial data, to evaluate their suitability for the purpose and to know the conditions applicable to their use' (European Commission, 2007). So, on many levels efforts have been made to pursue standardisation, however there is a need for a more general approach which would allow the finding and combining of all data and workflows available instead of only from the European Union on specific themes deemed most useful. Ma (2015), suggest a solution to these problems for the entire field of geoinformatics; the Semantic Web.

### Semantic Web

In accordance with Ma, Gil & Garijo (2012), with regards to making workflows more standardised and findable, and (Vilches-blázquez, Villazón-terrazas, Saquicela, & León, 2010) in the case of geo-data propose the same solution to the problem described above: The Semantic Web and Linked Data. The

Semantic Web can be defined as “ ... an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” (Berners-Lee, Hendler, & Lassila, 2001). Linked Data refers to the recommended best practices for exposing, sharing and connecting RDF data via dereferenceable Uniform Resource Identifiers (URI) on the Semantic Web. RDF is the Resource Description Framework, a way to describe information or data and is the standard used on the Semantic Web. The way data is described in RDF is through object – predicate – subject triples. The subject is the piece of data that is being described, the predicate describes a relationship, and the object is another piece of data. RDF describes data via dereferenceable URIs. The fact that they are dereferenceable means they can be looked up online providing more information on the specific resource. Other people online can link pieces of data to these same URIs creating a large network of data, the ideal is that all information online will be described in this way creating one large ‘Web of Data’. This would mean all (meta)data, workflows and analysis methods can be described using one standard. To create some structure on the Semantic Web, ontologies have been and are being developed. These ontologies describe what predicates can be used for what type of information, define classes, class hierarchy and class properties and create constraints. By using these well-defined ontologies, it allows computers to understand the relationship between different pieces of data and use inference to draw its own conclusions. The workings of the Semantic Web will be further explored in chapter 2.

#### A spatial problem-solving application

If we assume that the Semantic Web can offer a significant boost in standardisation, not only understandable by humans, but by machines also, we can imagine a semi-automated spatial problem-solving application. Figure 1.1 describes a possible system that takes all elements required in solving a spatial problem and interlinks them into one spatial problem-solving application with at its core a Semantic Web enabled agent.

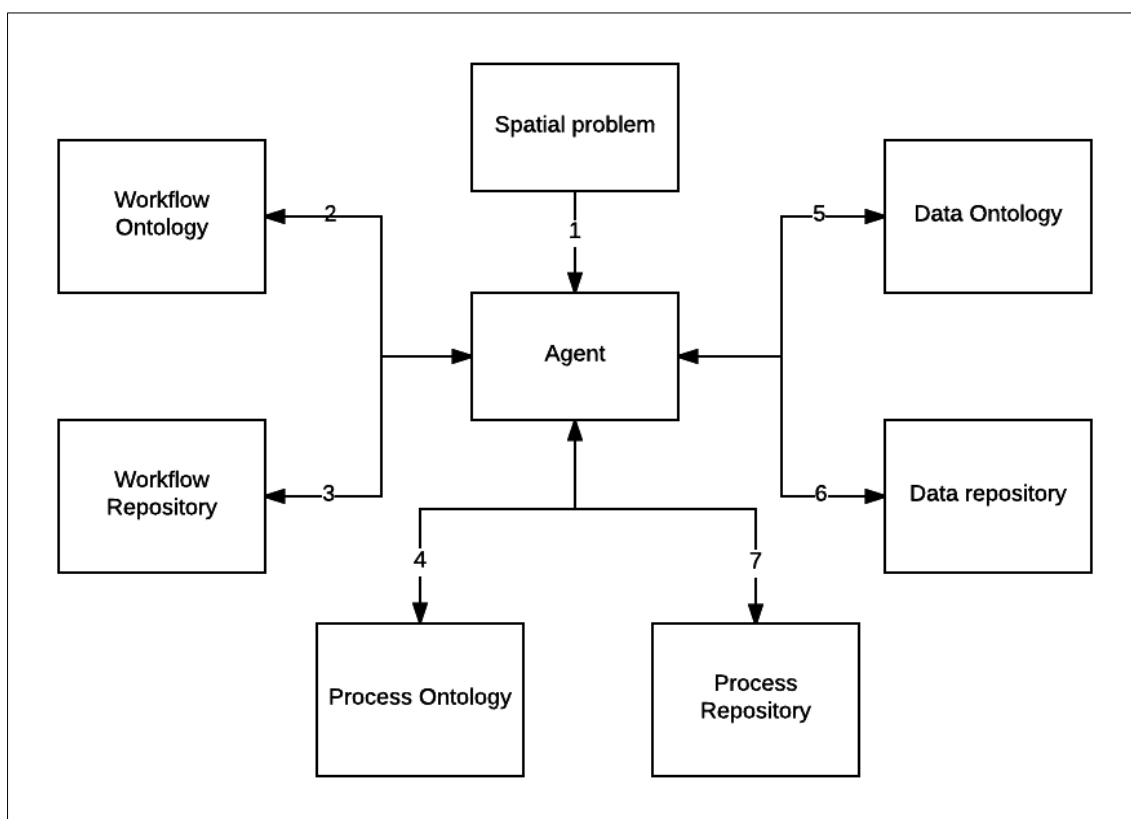


Figure 1.1: Spatial problem-solving application.

The imagined application goes through seven steps in order to solve a spatial problem. It starts with a user who has a spatial problem and asks a question, for example 'what crop should I grow on the land I just bought'. This question is asked to a central agent who will guide the user in obtaining all the required components needed to solve the spatial problem (1), for this some form of natural language processing will be required. The agent will then contact the workflow ontology, which describes the way in which workflows are described (2). Using this information, the workflow repository can be searched, which contains the methods to solve various spatial problems. Based on keywords, or other search facets suitable workflows can be searched for in this repository (3). A workflow repository contains the descriptions and locations of workflows all designed to solve a specific problem and will return one or more suitable workflows to the agent. The user can then read the workflow descriptions and analyse the proposed method and select the workflow most suitable to his problem. The agent will then obtain more information on the workflow by contacting a process ontology (4). An ontology is a standardised collection of descriptions of things and the relationship between these things. In this case this means that the process ontology provides descriptions of the different processes, its parameters and a description of the required input data. This extra information on the separate processes allows the user to make a better assessment of the method used and whether or not this method is the correct one to solve the problem at hand. If not, a different workflow can be selected. If so, further information on the parameters will be presented to the user in order to assist the user in providing the correct parameter values, if not already specified in the workflow. Otherwise, the agent will now contact a data ontology (5). This data ontology provides information on the data that is needed for the selected processes. In most cases part of the data needed will be described in the workflow. The workflow may specify that rainfall data is needed, for a workflow aiming to find the total rainfall in a certain period. The data ontology describes what different types of data exists, this specifies, for example, the collection method or the possible data collectors. Then the agent contacts the data repository (6). This data repository contains descriptions of datasets in a standardised way which corresponds to the terms used in the data ontology. In this way the selected data can easily be found and its location returned to the agent. The agent will look for the required data and return the descriptions of the dataset to the user. Based on this the user can now select the most relevant dataset. In some cases, this may already be specified in the workflow. Finally, the agent can contact the process repository (7) which contains an overview of execution engines which can execute the specified processes, these processes again are described using the terms specified in the process ontology. The agent can then provide a suggestion to the user where to execute the workflow, after the user selects the execution environment the agent will need to provide some form of translation of the workflow to a language the selected execution engine understands. The workflow, data and selected parameter values can then be send to the selected execution engine to obtain the wanted result.

Creating the application described above is of course easier said than done. It would have to be a system that is connected to a very large database and would require a far going level of standardisation in the field of geo-information and some form of Artificial Intelligence (AI) that allows the agent to assist the user in making the right choices throughout the process. In order to properly assess how utopian this idea is exactly a closer look has been taken at every step and what is needed in order to achieve this.

1. For the spatial problem to be understood properly by the agent, some form of natural language processing is required. For example, it should be able to understand that 'Ghana' or 'Utrecht' is a place and understand that 'crop' and 'grow' are keywords which should be matched with the workflow descriptions as provided by the creator of the workflow.

2. A workflow ontology is needed that describes a set of rules which allow a person to make the workflow findable for the agent. This will describe what a workflow is, what elements it contains, how to describe the purpose of the workflow and do all this with standardised terms that allow the workflows to be found and compared.
3. Some sort of search engine is required which the agent can access which finds the workflows based on the terms dictated in the workflow ontology. Workflows descriptions need to be matched to the key-words in the spatial problem stated by the user. These need to be returned to the agent.
4. Standardisation of operations is necessary to create the process ontology that is needed for this step. This consists of a classification of different types of operations, descriptions of all operations, the input and output parameters per operation and the methods per operation. This will require far going standardisation because currently the operation parameters and execution methods differ vastly between different execution engines.
5. The data ontology will need to provide the agent with the possible ways it can differentiate between data sets. This means that it should know what terms to look for in the metadata of the datasets in the data repository. To do this an overview needs to be made of possible collection methods, data collectors, collection intervals etc.
6. Similar to the workflow repository there needs to be some sort of search engine the agent can use to find data based on the information that has been obtained from the data ontology in combination with the datatype that has been specified in the workflow. In order to make the data findable, metadata should be constructed matching the structure and terms described in the ontology.
7. In order to be able to find the most suitable execution environment, a catalogue should be available which lists all possible execution engines (desktop clients, Web Processing Service (WPS) etc.) and the operations they can perform. This allows the agent to provide the user with a list of possible execution environments which are able to handle all operations in the workflow. Based on the user input gathered in the previous steps and the selection of the execution environment the workflow should be translated to a language the execution environment can understand.

## 1.2 Problem statement

The introduction of the Semantic Web as described above highlights a couple of elements of the Semantic Web that would make it the perfect platform for the proposed spatial problem-solving application. It uses ontologies, which through rules and constraints make an automatic standardisation effort. Furthermore, it is, obviously, online, which means that using the Semantic Web would allow for a universal system that everybody with internet access can use. Finally, and maybe most importantly, it allows for the computer to understand the elements of a workflow, the relationship between operations and its parameters and the descriptions of data. By understanding this it can actively suggest things like parameter values and eligible spatial data.

In theory the Semantic Web is the perfect solution to the problem, however research into this application of the Semantic Web is not yet, or just partially available. Therefore, this research will focus on applying Semantic Web technology to the spatial problem-solving application in order to come to a better understanding of the suitability of the Semantic Web for this purpose and the achievability of the application as a whole. However, as described before, a lot of effort is needed to create this entire application. Researching all seven steps and elements shown in Figure 1.1 is too large an effort to fit into the timeframe available for this thesis. Therefore, a selection has been made. The focus lies on the main goal of the application which is solving a spatial problem. The following steps of the application will be part of this research:

(3) sending the workflow to the agent – The standardisation of workflows and the creating of a workflow repository are outside the scope of this research, however discussing the state of the art is not. Either way, this step requires some form of reusable workflow as an input for the agent. This research discusses the possibility of using abstract workflows as an input, which the agent can then assist in converting to an executable workflow.

(4) process ontology – This research focussed on existing ontologies and where necessary enhanced or created them. The main focus here being the expediting of a general and standardised way to describe processes so they can be used in multiple workflows without confusion about the exact semantics. Furthermore, the description of the parameters per operation have been defined.

(7) conversion to execution – Where the earlier description of the application spoke of a catalogue of possible execution engines and the potential of distributed execution, that is outside of the scope of this research. The focus here lies with the possible methods of conversion and therefore focusses on a single offline execution environment.

These three parts of the spatial problem-solving application together allow the user to create or reuse an abstract solution to its problem, obtaining additional information on the methods using the Semantic Web (Semantic Enrichment), using this information parameter values for different operations can be determined and then the workflow can be converted to an executable format in a user-selected execution engine.

### 1.3 Research objectives

The aim of this research is the creation of a method which allows a user to come from an abstract workflow to an executable workflow using Semantic Web technology. This can be done by semantically enriching the workflow which then allows the user to obtain more information on the proposed methods and coming to a more concrete workflow which can then be converted to an executable workflow. The main research aim is: *The development of a method to semantically enrich an abstract workflow and convert it to an executable workflow using Semantic Web technologies.*

To be able to create such a method, several sub-objectives have been identified and corresponding question have been provided.

#### **Objective 1: Understanding the way the Semantic Web and Linked Data work.**

In order to be able to accurately assess the possible complexity of the proof of concept and scope of this research, first an assessment is necessary of the state of the art of the Semantic Web. This objective aims at gaining a sufficient understanding in order to do so.

##### *1.1 How are data sources connected on the Semantic Web?*

This question will provide insight into how Linked Data is actually linked. In answering this question knowledge will be gained on what the added value is of Linked Data and in what way this will allow for solutions that were not possible before.

##### *1.2 What tools are necessary to be able to use this technology?*

Here an overview will be created of the different tools/software/languages that are needed to make use of the Semantic Web. Whether these different tools are proprietary or open source will also influence whether they are suitable for this specific research.

### *1.3 What ontologies are available that could support the spatial problem-solving application?*

An essential element on the Semantic Web is the use of ontologies. This question will look into which ontologies are already available online, which of these are suitable for this research and in what way the existing ontologies can be enhanced in order to provide further support for the proof of concept.

### *1.4 How can ontologies/scripts/workflows designed during the research be stored/accessed on the Semantic Web?*

The objective of this question is to understand in what way data is stored and/or accessed on the Semantic Web. Is data that is currently available on the 'normal' web also accessible using Semantic Web standards, or do we have to convert such data first? What is the role of triple stores in this process? And how can we store data that is created during the creation of the proof of concept.

## **Objective 2: Assess different workflow design and scripting languages on their potential connectivity to the Semantic Web.**

Based on the knowledge gained in the answering of the questions in objective 1 an assessment can be made of the modelling tools and scripting languages that can be used for the creation of the conceptual and executable workflows. As the possibility to link these workflows to the Semantic Web will be partly based on the standards used to create them.

### *2.1 Which scripting languages, workflow design tools or standards are available for Geo Information (GI) workflows?*

To assess the best way to Semantically Enrich GI workflows, it is important to understand the possible tools and standards used to create them, both on the conceptual level as the executable level. For the executable level it is also relevant which GIS will be used for the execution of this workflow, as this will limit the possibilities to the languages the execution engine understands.

### *2.2 Do they allow for direct connectivity to the Semantic Web or is conversion or additional software needed?*

Possibly there is no way to create a GI workflow that can directly be used in coherence with Semantic Web standards. In this case to semantically enrich the workflow some conversion might be needed.

### *2.3 What software is suitable to construct the necessary workflows?*

Based on the answers to the previous question, suitable tools and standards are selected that provide easiest integration with Semantic Web standards.



**Objective 3: Assess the state of the art of the Semantic Web, to review the applicability of the selected use case to the proof of concept.**

To be able to assess whether the state of the art of the Semantic Web is advanced sufficiently to apply the selected use case to the proof of concept, a review is needed of the available ontologies to describe workflows and geo-processes.

*3.1 Are relevant ontologies available to describe the workflow and geo-processes?*

An overview is needed of the ontologies that are already available and whether enhancement of these ontologies suffices to successfully create the proof of concept.

*3.2 If ontologies are not available, is it possible to create own ontologies?*

If the existing ontologies are not sufficient to create the proof of concept a review should be made of available tools and standards to create a new ontology. This should provide insight into whether contributing to the Semantic Web is possible for users with limited experience, and if it is possible within the scope of this research.

*3.3 Is the complexity of the selected workflow sufficient to assess the added value of the Semantic Web?*

An assessment of the complexity of the used proof of concept workflow will provide some additional insight into the success of the application. For example the difference in types of operations, number of operations and amount and type of data used will say something about the complexity of the workflow used.

*3.4 Who are the expected users of the proof of concept?*

This question will look at who might be the potential users of the proof of concept as during the creation of the proof of concept, the users should be kept in mind with respect to the usability and complexity of the system.

**Objective 4: Assess the creation process of the proof of concept and its success.**

The process of creating the proof of concept will be elaborated on in chapter 4. The questions here serve as guidance in assessing the success of the proof of concept and the influence choices made have had on the proof of concept.

*4.1 How did the selected tools and standards influence the proof of concept?*

Objective 1 and 2 focus on selecting the appropriate tools and standards to create the proof of concept, it is important to reflect on these choices in order to assess their contribution and suitability for this project.

*4.2 Did the proof of concept provide relevant output to assess the success of the proof of concept?*

The output of the proof of concept should be twofold, an executable workflow, as well as a result obtained by executing this workflow. These outputs allow the assessment of the success of the proof of concept.

#### *4.3 How can the proof of concept be tested with probable users?*

To create a useful proof of concept, it is important to keep in mind the end-user. In this case the problem-solving application should be able to assist non-expert users in solving a spatial problem. Keeping in mind the possibility to test the proof of concept with probable users, allows for easier follow-up research.

### **Objective 5: Analyse the results of the proof of concept and provide recommendations towards further research.**

This objective and its questions have been designed to pinpoint the exact shortcomings of the current system and the cause of these shortcomings. By being able to accurately describe the issues at hand it is possible to provide useful recommendations for further research.

#### *5.1 Which elements of the system work as intended and which should be improved?*

This question will provide insight into whether the results are as to be expected. If they are not it is important to determine which step of the process is not working correctly and why.

#### *5.2 Are any limitations caused by the selected tools or are there issues with the underlying technology?*

If the proof of concept does not function as would be expected, it is important to determine why certain steps have failed. Basically, there are two options, the tools that have been selected are not working correctly, or the combination of tools does not work. Otherwise, the technology being used is not advanced enough or not the right fit for this type of application. This question will provide insight to what may have caused the issues.

#### *5.3 What steps are necessary to apply this system on a larger scale or within organisations?*

Based on the answers to the previous two questions, the answer to this question will shed some light on how far the technology has developed and what steps are still necessary to be able to apply this method on a larger scale. Here efficiency, ease of use and reliability play an important role.

### **1.4 Use Case**

As a use case for this thesis the MaMaSe project has been selected. The MaMaSe project, or in full, the MaMaSe Sustainable Water Initiative, *'is aimed at improving water safety and security in the Mara River Basin to support structural poverty reduction, sustainable economic growth and conservation of the basin's ecosystems'* (Mamase.org, 2016). There are two reasons for selecting this use case. First, a suitable use case for this research project entails a project in which spatial problems are solved of which the reuse of the workflows (by non-expert users) has a clear benefit, as this proves the usefulness of the proposed application. Furthermore, to direct all attention towards the research itself and not the preparation of the workflow and its data, a use case has been selected of which the workflows and the data is readily available. Both these requirements are met with the MaMaSe

project. As the supervising institution, the ITC, is involved in the MaMaSe project they can provide access to the workflows and data. The usefulness of reusing and sharing the workflows of this project are also apparent. The questions answered by the workflows in this project and the knowledge gained is obviously very useful for other regions in less developed countries. Therefore, this use case is very suitable for this research and it has been used to select an example workflow from to test the created application.

### 1.5 Research Limitations

Research into the Semantic Web, and the implementation of it on a large scale is still very much in a preliminary stage. Even though more and more fully developed tools are available, many are still experimental, with limited documentation. The goal of this research is to prove the suitability of the Semantic Web as an integral part of solving spatial problems. This means that rather than testing with a broad range of functions and workflows the focus is on proving the conversion from abstract to executable is possible, supported by a geo-operation ontology and Semantic Web standards. Therefore, this research will focus on one proof of concept workflow with a subset of geo-operations. It in no way intends to set new standards, or provide full ontologies, ready to be implemented on a large scale. Furthermore, user testing is kept in mind during the research, as a focus on the end user provides guidance in creating a useful prove of concept. However, actual user testing is something that should be part of future research, and will not be attempted in this thesis.

### 1.6 Reading guide

The next two chapters provide an overview of related work and go into more detail about the Semantic Web and workflows. The chapter on the Semantic Web will provide some insight into the inner workings of the Semantic Web, and showcases the necessary information for the reader to understand how it will be applied to the proof of concept. The chapter on workflows looks at different types of workflows, how they are currently created and managed, the sharing of workflows and related work on workflow abstractions and the use of Semantic Web. After the theoretical chapters, chapter 4 looks at how the proof of concept was created, and what design choices were made. After this a analysis of results chapter will follow describing the knowledge obtained while creating the proof of concept and providing a reflection on the selected methods and issues tackled during the creation of the proof of concept. The thesis ends with the conclusions that can be drawn, the answering of the research questions and recommendations for further research in order to take the next step in using the Semantic Web as a spatial problem solver.

## 2 Semantic Web

The Semantic Web, first introduced by Tim Berners-Lee in 2001, has been described as “ ... *an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*” (Berners-Lee et al., 2001). The most important aspect being the well-defined meaning, which cannot only be understood by humans but also by computers allowing them to work together. In the early stages this was mainly a utopian concept without a clear idea about the feasibility of this idea. However, the introduction of Linked Data, which is the main technology behind the Semantic Web, provided a means to an end. Bizer, Heath & Berners-Lee, some of the driving forces behind Linked Data described it as follows: “*Technically, Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external datasets and can in turn be linked to from external datasets*’ (Bizer, Heath, & Berners-Lee, 2009).

### 2.1 Linked Data

The most important aspect in the reuse of data is a well-defined structure. This allows other people and computers to understand the semantics of a dataset and the way it relates to other aspects in this or other datasets. However, even in this information age, the main language on the web, HTML, is only focussed on providing structure to text, rather than data. Even though methods have been devised, working with micro formats, to provide some structure and to automatically extract information from an HTML document, this leads no further than, for example, understanding that a certain name and a book are related. The relationship between the name and the book is lost, so it might be a character in the book but it could also be the author. A way to provide structured data on the web is the use of an Application Programming Interface (API). These provide requested data in standardised formats such as XML or JSON, formats that can be parsed by most programming languages and allow the reuse of this data. However, data in APIs tends to be structured by a local data model, using identifiers with a local scope. This means that as soon as some data is extracted and taken out of the context of the dataset it loses its meaning and the semantics become unclear. This is where Linked Data comes in, it proposes a method of linking data from different datasets to each other by suggesting a standard data model, allowing everyone to publish their data on the web, in a similar fashion as is currently being done with HTML documents on the World Wide Web (Heath & Bizer, 2011). This would allow the web to evolve from a ‘*web of documents*’ to a ‘*web of data*’. In order to make certain that everybody published their data in a standardised way Tim Berners-Lee created a set of rules known as the Linked Data principles, allowing all published data to become part of a single global data space (Bizer et al., 2009):

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things

Before going into the first two principles, explaining how URIs can be used to identify real world things, a look will be taken at principle three, how to use the RDF standard to provide useful information to people who look up a specific resource.

### 2.2 Resource Description Framework

The Resource Description Framework (RDF), a data model that represents a semantic network, is a directed network with qualified edges. A regular directed network is homogenous as the edges are represented as an ordered pair, and the edges are of an unspecified type of relationship. So this relationship is similar for the entire network (e.g. a friendship network). RDF, however, makes use of

qualified edges, which allows a network to be heterogeneous in nature, as every edge can have a different meaning (Rodriguez, Bollen, Gershenson, & Watkins, 2012). This allows for the description of a book having an author, but also having a main character, and having a publisher. This is done using so-called subject – predicate – object triples. These triples can also use the described resources as object as well as subject. In one RDF document a triple can be book – hasAuthor – Stephen King and the next triple can be Stephen King – hasBirthPlace – Portland (Figure 2.1). As the resource Stephen King is first an object and then a subject this creates a graph like structure of resource being linked to each other in one RDF document. This is what the fourth Linked Data principle refers to, by including links to other resources in an RDF document this allows the linking to different datasets that are not under your control in order to create a global Web of Data, and turning the internet into one large knowledge graph where all information is linked to each other. So how can we identify these different resources in such a way that they are unique and are also reusable by others?

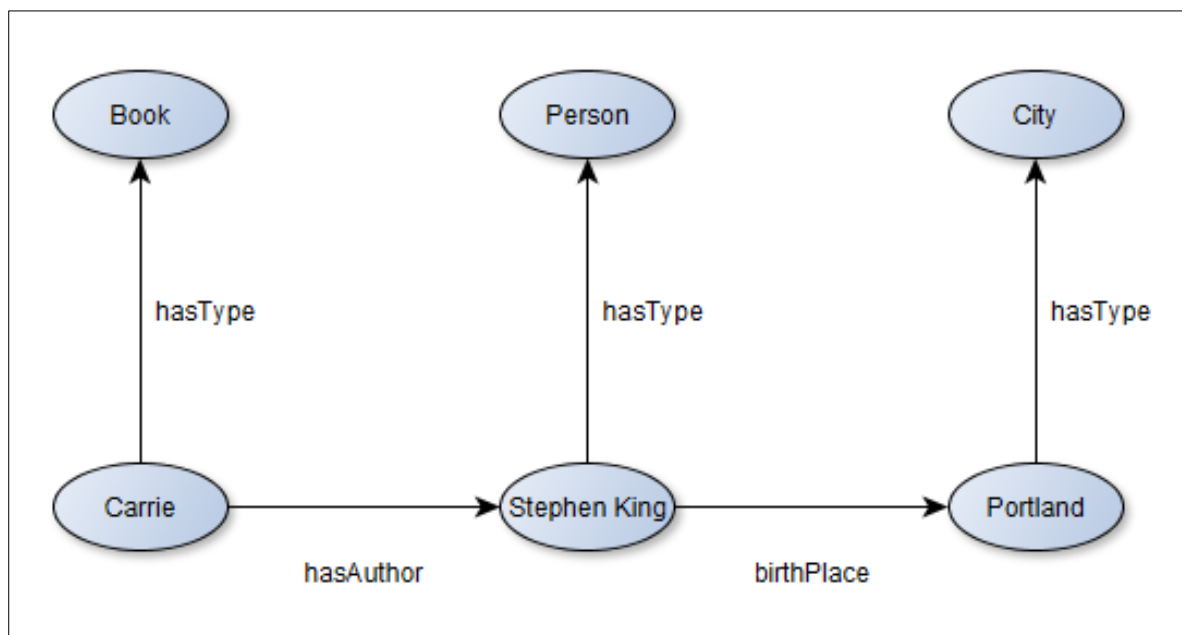


Figure 2.1: Example of how RDF structure forms a graph.

### 2.2.1 Uniform Resource Identifiers

The first Linked Data principle describes the use of URIs as names for things. URIs are Uniform Resource Identifiers, which not only serve as a unique identifier for a resource but also describe the location of the resource on the web of data. URIs come in different formats, the most well-known being the Uniform Resource Locator (URL) which is used to identify websites. URIs mostly consist of two parts, being the identifier of the resource and the domain in which it is located. For example, the URI of the author Stephen King could be ‘example.org/writers/Stephen\_King’. So here the first part of the URI ‘example.org’, describes the domain, more commonly referred to as the namespace in Linked Data, and the second part ‘writers/Stephen\_King’ provides a unique identifier in that namespace. The namespace in this case often describes the data owner or publisher, or the dataset the resources belong to. The unique identifier describes the resource, and only has to be unique in that specific namespace. Even though URIs are used to describe resources they are not limited to describing only online resources. They are also used to describe real world entities such as humans, books or even concepts such as love, or peace. This means that URI describing an online store would not be the URL of the homepage of that store, but a separate URI describing the real-world entity that that store

represents. The URI for amazon for example could be ‘example.org/webstores/amazon’ rather than ‘www.amazon.com’. The latter would be a resource providing more information on the former resource using the predicate ‘hasWebsite’.

For people to be able to look up these URIs and be provided with more information on these resources, the second Linked Data principle refers to using HTTP URIs. HTTP is the protocol currently used on the internet to communicate between a web browser and a webserver. In most basic terms this means that every URI should start with http://, however it also means that the web of data makes use of the architecture of the current web of documents. Meaning the two can co-exist and the web of data is easily accessible to all users of the current web. On the technical side it also means that content negotiation is possible, which is part of the http protocol. This allows the webserver to understand who is requesting the information and providing the resource in the right format. This means that for computers and humans the same URI can be used for a resource, but depending on who is requesting the information either an RDF representation or an HTML representation will be served.

When creating Linked Data, the URIs used to describe resources are very important. As people will start linking their data to the resources you describe it all comes down to creating persistent URIs, so URIs that never change. If in a later stage the URIs used would change this would leave all created links to those resources invalid [1]. Important aspects in developing a URI strategy are having control of the namespace in which they are minted, using unique universal identifiers (e.g. ISBN numbers for books), and leaving implementation details out (e.g. server names, portnumbers) as these are often subject to change (Heath & Bizer, 2011).

### 2.2.2 RDF syntax and serialisation

When converting the Stephen King example provided above to URIs it looks like Figure 2.2. As can be seen most of the resources are in the DBpedia namespace, this is a Linked Data version of Wikipedia. DBpedia is a ‘crowd-sourced community effort’ that is aimed at taking structured information from Wikipedia and allowing people to ask sophisticated queries, and linking external datasets to Wikipedia and gain new insights into this rich information resource (DBpedia.org, 2017). Furthermore, the RDFS and FOAF namespaces can be found. These are commonly used namespaces, and will be discussed in

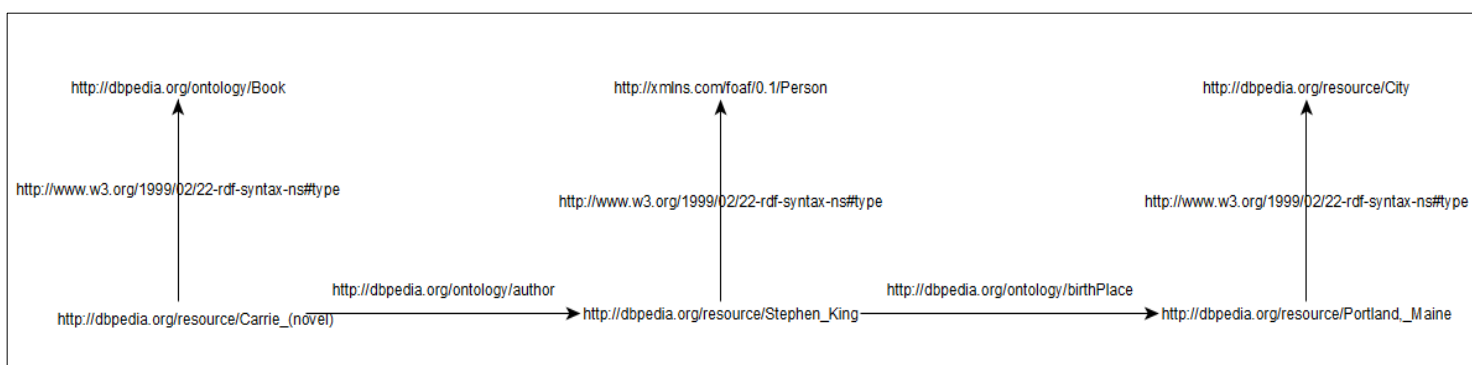


Figure 2.2: RDF example using URIs.

more detail in section 2.3. This section will focus on discussing some of the relevant RDF syntax and serialisation. As Linked Data described in an RDF document does not come in the form of a graph but rather, in the original format RDF/XML, it is less easily readable than figure 2.2. However, there are several other serialisations of RDF that improve readability for humans. One that will be used throughout this thesis will be discussed as this will improve the understanding of the examples provided later on.

The RDF/XML syntax (Listing 2.1) is the original syntax of RDF, when it was first developed at the end of the last decade. The most relevant tags here are the `rdf:RDF` start tag, where short hand is defined for common namespaces, in this case `dbo` for the DBpedia ontology, and `rdf` for the `rdf` syntax namespace. These can then be used throughout the document as a replacement of the full namespace. Furthermore, the syntax is based on the `rdf:Description`, `rdf:about`, and `rdf:resource` tags. The description tag surrounds the entire triple, where `rdf:about` describes the resource that is the subject of the triple. Then tags describing the predicate are used, with `rdf:resource` to specify the object of a triple (W3C, RDF 1.1 Primer, 2014).

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:dbo="http://dbpedia.org/ontology/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://dbpedia.org/resource/Carrie_(novel)">
    <rdf:type rdf:resource="http://dbpedia.org/ontology/Book"/>
    <dbo:author rdf:resource="http://dbpedia.org/resource/Stephen_King"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/resource/Stephen_King">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <dbo:birthPlace rdf:resource="http://dbpedia.org/resource/Portland,_Maine"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/resource/Portland,_Maine">
    <rdf:type rdf:resource="http://dbpedia.org/resource/City"/>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.1: Example of RDF/XML serialisation.

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://dbpedia.org/resource/Carrie_(novel)> a dbo:Book ;
  dbo:author <http://dbpedia.org/resource/Stephen_King> .

<http://dbpedia.org/resource/Portland,_Maine> a <http://dbpedia.org/resource/City> .

<http://dbpedia.org/resource/Stephen_King> a <http://xmlns.com/foaf/0.1/Person> ;
  dbo:birthPlace <http://dbpedia.org/resource/Portland,_Maine> .
```

Listing 2.2: Example of turtle serialisation.

The `rdf` serialisation that will be used throughout this thesis is turtle (Listing 2.2). This serialisation ‘provides a trade-off between ease of writing, ease of parsing and readability’. This serialisation has been selected for the readability aspect, as it is easy to understand for humans. Similar to the shorthand provided in the XML example, the turtle syntax uses prefixes to define shorthand for the often-used namespaces. This means that `dbo:Book` on line 7 actually represents `<http://dbpedia.org/ontology/Book>`. Another piece of enhanced readability is provided by `a` as shorthand for `rdf:type`. `rdf:type` is one of the most used predicates in `rdf` documents as it describes the class of a certain entity. Using the letter `a` as a shorthand for this, means line 7 actually reads:

Carrie is a Book. Furthermore, turtle allows the use of ; to indicate the next triple uses the same subject, as can be seen at the end of line 7. It is also possible to use a comma, to indicate the subject and predicate are the same, only specifying a different object (W3C, RDF 1.1 Primer, 2014).

A third serialisation that is relevant to discuss is the JavaScript Object Notation – Linked Data (JSON-LD), a serialisation that is easily understood by machines, as it can be parsed in the same way as JSON. JSON is a popular format to store data used in web-applications. Serialising RDF data as JSON-LD allows developers to use it as regular JSON but with the elective benefit of added semantics if they choose to use this. This means that the same dataset can be used for both Semantic Web as regular web applications without any need for conversion. As with the XML and turtle examples, JSON-LD uses prefixes to allow shorter notation of the URIs throughout the document (Listing 2.3). The prefixes here are defined as @context at the start of the document. After this a JSON object is created referring to resources using @id, and specifying their type by using @type. Other predicates than rdf:type are simply added to the array without @ and again use @id to specify the resource that is the object of the triple (W3C, JSON-LD Primer, 2017).

```
{
  "@context": {
    "dbo": "http://dbpedia.org/ontology/",
    "dbpedia": "http://dbpedia.org/resource/",
    "foaf": "http://xmlns.com/foaf/0.1/",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "dbpedia:Carrie_(novel)",
      "@type": "dbo:Book",
      "dbo:author": {
        "@id": "dbpedia:Stephen_King"
      }
    },
    {
      "@id": "dbpedia:Stephen_King",
      "@type": "foaf:Person",
      "dbo:birthPlace": {
        "@id": "dbpedia:Portland,_Maine"
      }
    },
    {
      "@id": "dbpedia:Portland,_Maine",
      "@type": "dbpedia:City"
    }
  ]
}
```

Listing 2.3: Example of JSON-LD serialisation.

## 2.3 Ontologies

Section 2.2 already mentioned the RDF and DBpedia ontology. This section will provide more insight into ontologies, their role in the structure of the Semantic Web and the way they are modelled. A fundamental reason for using ontologies is to avoid the ambiguity and vagueness of natural language. In an ontology the meaning associated with data is well-defined using markup that facilitates disambiguation and defines structure (Ciocoiu, Gruninger, & Nau, 2000). So, ontologies are formal descriptions of the concepts, terms, and relationships within a given knowledge domain. As the RDF standard only provides the structure of the data but does not provide any domain specific terms to describe classes of things or URIs for the predicates that describe the relationship between entities, this is up to the domain experts themselves. This is done in ontologies described in one of the ontology languages; Simple Knowledge Organization System (SKOS), RDF Schema (RDFS) or the Web Ontology Language (OWL) (Heath & Bizer, 2011). All ontology languages are based on RDF, using this standard to model the ontologies. A full description of these languages goes beyond the scope of this thesis, but as different ontologies have been used in the development of the proof of concept, a basic understanding of the different aspects of these languages is required, to understand the way these ontologies have been used.



## Modelling ontologies

To understand the way an ontology has been modelled it is important to understand what the different ontology languages have to offer. The SKOS vocabulary is mainly focussed on semi-formal knowledge organization systems (thesauri, taxonomies) and it describes concepts rather than specific entities. As this language misses the hierarchical and formal nature that is required for this particular research, it will not be discussed in any further detail (Allemang & Hendler, 2008). The RDF schema or RDFS is basically the backbone of RDF. It consists of 8 classes and 7 properties, which allow the description of any resource in RDF. The most relevant will be discussed.

Everything that is described by RDF is an instance of the class `rdfs:Resource`, this means it is the main class in RDF, all other classes are subclasses of `rdfs:Resource`. The other class in the RDF schema that is relevant to this thesis is `rdfs:Class`, which allows the instantiation of classes when developing an ontology. As every resource in an RDF document has a specific type and thereby belongs to a certain class, this is of great importance when modelling data in RDF. The next element when modelling an ontology is defining properties which the instance of a certain class can have. All properties belong to the class `rdf:Property`. All instances of `rdf:Property` have the `rdfs:range` and `rdfs:domain` properties describing for which classes they are valid. Properties basically describe the middle, or predicate part, of a triple. The range of a property defines to which class the object of a triple belongs, where the domain of a property describes to which class the subject of a triple belongs. If we take the example earlier *Carrie – author – Stephen King*, the domain of the author property is `dbo:Book` and the range of the author property is `foaf:Person`. What these properties do is basically restrict people who use this ontology from using these properties on instances of different classes, providing some basic logic to the dataset. Further relevant properties in the RDF Schema are `rdf:type` specifying the class an instance belongs to and `rdfs:subClassOf` specifying hierarchy in classes. Finally, RDFS defines `rdfs:label` and `rdfs:comment`, these properties are meant to provide a natural language description and label to resources. This allows an application that grabs RDF data from the Semantic Web to provide users with a plain text label and description of the resource instead of a URI, allowing for better usability and readability (W3C, RDF Schema 1.1, 2014).

Where the RDF Schema provides some backbone of classes and properties the OWL language provides more logic and restrictions and provides an ontology defined in RDFS with some extra expressivity. There is some overlap between the two ontology languages, OWL for example also provides the `owl:Class`, but in many cases they are complementary. For example, OWL provides `owl:equivalentClass` and `owl:equivalentProperty`, these properties allow the mapping between two different ontologies by linking classes and properties (Heath & Bizer, 2011). Further logic in the ontologies is provided by the `owl:AllDisjointClasses`, specifying that a member of a certain class cannot be the member of a certain other class. So, an instance of the class `Man` cannot also be a member of the class `Woman`. OWL also provides some more specificity in certain cases, it, for example, has an `owl:ObjectProperty` which always has a resource as a range. On the other hand it has an `owl:DatatypeProperty` which always has to specify a datatype as range such as string or integer. (W3C, OWL 2.0 Primer, 2012).

## 2.4 SPARQL

The third Linked Data principle specifies two standards that should be used when providing information on resources on the Semantic Web. First of all, RDF, which has been explained above, and secondly SPARQL. SPARQL, or Simple Protocol and RDF Query Language, was first standardised in 2008, and its most recent version SPARQL 1.1 was standardised in 2013. SPARQL allows the querying of RDF knowledge graphs that are exposed through a SPARQL endpoint. SPARQL makes use of triple patterns similar to those described in RDF documents however it allows for the replacement of some of the resources by variables (W3C, SPARQL 1.1). SPARQL supports four different types of queries:

SELECT – This returns all, or a subset of the variables that are bound in the query pattern specified. This returns raw data in the form of a table.

ASK – An ASK query returns a boolean that indicates whether the specified triple pattern matches something in the dataset or not.

CONSTRUCT – The CONSTRUCT query grabs data matching the specified triple pattern, however it returns it in the form of an RDF graph constructed by substituting variables in a set of triple patterns specified in the query.

DESCRIBE – IT returns an RDF graph that describes the resources that have been found. Specifying a specific triple pattern to look for is optional.

Listing 2.4 shows an example of a SPARQL SELECT query. It consists of a prefix declaration, a select clause, a where clause and optional query modifiers. The select clause describes which variables should be returned in the result set, so in this case all the books written by Stephen King. The where clause specifies where it should find these books. Finally, the query modifier, in this case a limiter, says it should only return the first 50 results. Other possible query modifiers are filters and grouping and sorting operators.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?books WHERE {
  ?books a dbo:Book;
         dbo:author <http://dbpedia.org/resource/Stephen_King> .
}
LIMIT 50
```

Listing 2.4: Example of a SPARQL select query.

## GeoSPARQL

In order to be able to extend queries on Linked Data with the possibility to ask questions about topological relations the GeoSPARQL extension was developed by the Open Geospatial Consortium (OGC). It's an attempt to link the Geo oriented standards of the OGC to the Semantic Web. It provides a vocabulary that allows to describe geographic data and furthermore provides different sets of topological query functions which can be used in queries that request this geographic data. The sets of topological functions are the Simple Features, Egenhofer and RCC8 spatial relations (OGC, 2012). Finally, it also provides a set of basic geographical operators such as buffer, intersection and union. In the context of workflows these simple operators and topological relations might be able to perform part of the tasks of the workflow in the process of querying the data, making the process more efficient.

## 2.5 Semantic Web Tools

As Berners-Lee already introduced the idea of the Semantic Web in 2001, there has been more than enough time to develop all sorts of tools that play some part in it. Ranging from commercial vendors, to experimental or research-based tools, and from very successful to underdeveloped prototypes. This section provides an overview of tools that can be used in creating, exploring, querying and visualising Linked Data. These have been categorized by their function, and have been tested to assess their usefulness for this thesis and are all still functioning. Every section will provide a short explanation of the use of such a tool, and an overview of some of the current best options.

### 2.5.1 Triple Stores

Triple stores are the databases of the Semantic Web. As the name already states they store triples, or in other words, RDF graphs. There is a very large range of different triple stores varying from native triple stores, to relational databases with extension to be able to deal with RDF data. Also, when it comes to their handling of SPARQL it varies from a full implementation of the SPARQL standard, to triple stores that, under the hood, translate the SPARQL to SQL queries. As this research focusses on the use of Geo Information workflows, only triple stores that support GeoSPARQL queries have been selected.

*Parliament* - Parliament has an almost complete implementation of the GeoSPARQL standard. Next to this it is further compatible with RDF, RDFS, OWL and provides a SPARQL endpoint that can be queried from outside of the database. It allows querying over indexed geometries using a standard R-tree. It makes use of a resource table, statement table and a dictionary for research mapping, which allows it to reorder queries to do the most selective parts first. By splitting the queries that come in it allows for efficient query planning leveraging the spatial and thematic components (Patroumpas, Giannopoulos, & Athanasiou, 2014)

*GraphDB* – GraphDB is a triple store designed for medium data volumes, up to 100 million triples. It includes an implementation of OpenRefine which allows the user to refine RDF datasets before adding them to the store, but also turning tabular data into RDF. It doesn't naturally allow GeoSparql queries, but a plugin that comes with the download of GraphDB does create this possibility. It also has a highly developed GUI that makes it very easy in its use and is, next to a free version, also available on a commercial license, providing round the clock support. This makes it one of the more commercially developed triple store, however, little research is available on the performance of GraphDB compared to other triple stores.

*Virtuoso* – Virtuoso is a popular triple store, which among others, provides the SPARQL endpoint for DBpedia. In contrast to the other triple stores discussed here it uses a DBMS back-end. One of the advantages of Virtuoso is that it is using a caching mechanism meaning that if the same query is run more often it decreases in runtime. However, query runtimes in Virtuoso are rather skewed with relatively high standard deviation (Schätzle, Przyjaciel-Zablocki, Skilevic, & Lausen, 2015). When it comes to GeoSPARQL Virtuoso is not fully compliant with the standard, but incorporates a large part of the possibilities.

### 2.5.2 SPARQL builder

SPARQL builders are tools that assist in creating SPARQL queries. Not only is little knowledge of SPARQL required to use these tools, in most cases also little knowledge of the dataset that is being queried is required. This can support in easier understanding of often complex RDF datasets. Two of these tools are discussed below.

*Sparklis* – Sparklis is a tool that is based on natural language to guide the user in building SPARQL queries. It incorporates a lot of the possibilities of the SPARQL standard (e.g. union, filters, aggregation). By using natural language sentences in either French or English, no further knowledge of vocabularies or data schemas is needed in order to create full SPARQL queries. However, it does require that OWL or RDFS standards have been applied, in order to discover the structure of the data on its own (Ferre, 2017).

*LinDA* – LinDA is a drag-n-drop query designer building on the query designers known for SQL and relational databases. When pointed to an endpoint it automatically retrieves the classes and properties, allowing the user to simply select the required classes from a drop-down menu. When

classes have properties pointing to another class, the user can traverse through the virtual RDF graph and on the fly SPARQL is created. As with Sparklis no prior knowledge of the SPARQL language is needed to use LinDA. LinDA does require the use of standard class description using OWL or RDFS. Furthermore, a basic understanding of triple patterns is required to use LinDA.

### 2.5.3 Visualisation tools

Another way of gaining a better understanding of an RDF dataset are visualisation tools. Rather than using SPARQL to query resources and finding the core elements of a dataset, these tools visualise the triples patterns in various ways which provide a different view which may lead to other insights into the data.

*SPEX* – Spex is a spatio-temporal content explorer, which allows exploration of SPARQL endpoints of which the content or at least the schema is unknown. It provides several views on the data, among which a map view, a timeline, a result set window and the SPARQL query that is currently being sent to the endpoint. The main window provides a variable which can be set to a specific class in the endpoint, from here filtering can take place on the map view (if geometries are provided) or the timeline (if dates are provided). Alternatively, properties can be selected linking to other classes, to further limit the result set, finding specific entities of interest (Scheider et al., 2017).

*Gruff* – Gruff is a visual triple store browser. It visualises the graph, and by displaying tables of properties, colour coding, and an overview of classes it provides an alternative view on the data. It provides the possibility to grab a random subset of the data from an endpoint and start exploration from there. Clicking a certain resource, quickly shows all resources linked to it, and their predicates, and from there the user can traverse through the graph, providing a table of with the history of previous nodes clicked. It is also possible to start exploration from a self-defined point, by either specifying the URI or the rdfs:label, by selecting a rdf:type or using free text-lookup among all properties values containing natural language (Aasman & Cheetham, 2011).

*WebVOWL* – WebVOWL is an online tool focussed on the visualisation of OWL ontologies. These can be uploaded in the form of OWL, for which a converter is available (OWL2VOWL). As ontologies are often complex structures of class hierarchy and properties, a visualisation tool can assist in making sense of an unknown ontology. WebVOWL provides a graph based view of the ontology, showing classes with its properties and how classes are related, a sidebar provides information on things such as title, description, namespace, authors and version. Via drag n drop the nodes can be rearranged to provide different views. Furthermore, there are filters for datatypes, and solitary subclasses (Lohmann, Link, Marbach, & Negru, 2014).

*Protégé* – Protégé is not only a way to more easily understand OWL ontologies, it rather is an ontology builder. It provides the tools to easily create an ontology without resorting to writing it in OWL in a text editor. On top of allowing easy creation of classes and properties, its strength lies in the build-in reasoner, which verifies the logic of the ontology. Furthermore, it infers super classes, and checks the logic behind OWL constructs such as disjoint classes. Finally, it also provides several plugins which, among other features, provide a wide range of different visualisation tools.

### 3 Workflows

Workflows have been around for several decades and have gone through series of standardisation and change over time. Since 1993 the WorkFlow Management Coalition (WFMC) has worked to create standards focussing mainly on standardisation of business processes. In 1996 they defined workflows as: *'the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules'* (Barga & Gannon, 2007). This definition, as many that are found when defining the term 'workflow' specifically refer to business in the definition. Opposed to this are the scientific workflows which have more recently emerged as a new paradigm (Lin et al., 2008). The definitions used to describe scientific workflows have clear differences with their business counter parts: *"Workflows capture the individual data transformations and analysis steps as well as the mechanisms to carry them out in a distributed environment."* (Gil et al., 2007). Where business workflows discuss documents, information and tasks, the scientific workflow refers to data transformation and analysis steps. Furthermore, the tasks in business workflows are carried out by participants where in the scientific world these are carried out by *'mechanisms in a distributed environment'*. Even though in more modern definitions of business workflows there is more room for computer tasks and far going automation, they still leave room for human tasks as not everything can be fully automated (Sonntag, Karastoyanova, & Deelman, 2010). Furthermore, a characterizing element of business workflows which differentiates it from the scientific workflows is the need for integrity. Usually a business workflow implements a company's product or service. Robust execution is therefore essential as the customer is paying for a specific outcome. Scientific workflows on the other hand might itself be subject of research, and failure halfway through execution might be an accepted outcome (Barga & Gannon, 2007; Sonntag et al., 2010)

As this thesis focusses on the use of workflows that consist solely of computer tasks, as this makes them useable in communities where domain experts are absent, the focus will lie with scientific workflows rather than business workflows. This means no human-tasks are involved, except for the creation of the workflow. The definition used for this type of workflow is a workflow which 'captures the individual data transformations and analysis steps as well as the mechanisms to carry them out'. The original definition also mentioned a distributed environment, however, this research will use a local execution environment as using a distributed environment would add complexity and effort without the immediate reward of a more intricate research outcome. Every step in such a workflow refers to a process or computation that has to be executed (e.g., web-service to be invoked, or a calculation to be performed). The logic of these workflows consists of on an execution order of the different steps based on the data flow and dependencies among the different elements. The reason for the widely adopted use of workflows lies in the ability of workflows to 'capture complex analysis processes at various levels of abstraction, and also provide the provenance information necessary for scientific reproducibility, result publication, and result sharing among collaborators' (Gil, Deelman, et al., 2007).

This chapter will provide insight into advantages of scientific workflows, but also the existing issues surrounding sharing and reusing workflows. Different abstraction methods of workflows will be introduced which allow more efficient sharing of workflows. Furthermore, an exploration of related work involving the Semantic Web will be presented, followed by a section shedding some light on how the Semantic Web can improve the way the community deals with workflows. Finally, a second look shall be taken at how the Semantic Web and abstract workflows can improve the introduced spatial problem-solving application.

### 3.1 Scientific Workflows

A scientific workflow is an explicit protocol that captures the, often tacit, knowledge of a research team. It allows for the sharing of not only the results of an experiment but also the methods in an explicit and reusable way. The main advantages of the use of scientific workflows can be categorised into two aspects. First, they serve as detailed documentation of the research methodology, and basically provide all provenance information required to assess the data that is produced by the workflow. Secondly, it is a reusable artefact that facilitates in two of the fundamentals of science, the reproducibility and repeatability of scientific research. The former facilitating the latter, as the degree of re-usability is defined by the way in which the workflow is created, described and shared.

#### Re-use of workflows

Using workflows created by others can be done with three different goals, repeating the steps that have been taken in an experiment with altered settings or variables, reproducing scientific work to evaluate results, and re-using part or all of the workflow in order to produce new or more elaborate results. Repeatability is concerned with repeating an experiment in the same setting to assess the reliability of the results. Reproducibility of scientific analyses and processes, which can be seen as a special case of repeatability, is a key requirement to assess the credibility of novel claims. The use of workflows in these processes does not only allow the possibility to reproduce the final results in order to evaluate the validity of someone's hypothesis, but also allows verifying intermediate results (Bechhofer et al., 2013). The reuse of workflows in the sense of incremental/evolutionary workflow development allows for a more efficient scientific process, where one builds upon the progress achieved by others (Garijo et al., 2014). As mentioned before the repeatability, reproducibility and reusability of the workflows are facilitated by the way in which workflows store provenance information and tacit knowledge allowing for a clear assessment of the data transformations and methodological steps.

#### Provenance

Provenance has a lot of different definitions across different domains. It can differ between data and process oriented provenance, and can be stored at different levels of detail. Also, depending on the use of a workflow, it can be used in order to determine different things, among which determining the data quality, data transformation methods, replication recipes and attribution (Simmhan, Plale, & Gannon, 2005). One of the provenance standards often used is PROV, to get a better understanding of what information a provenance model stores see Figure 3.1 taken from the PROV standard

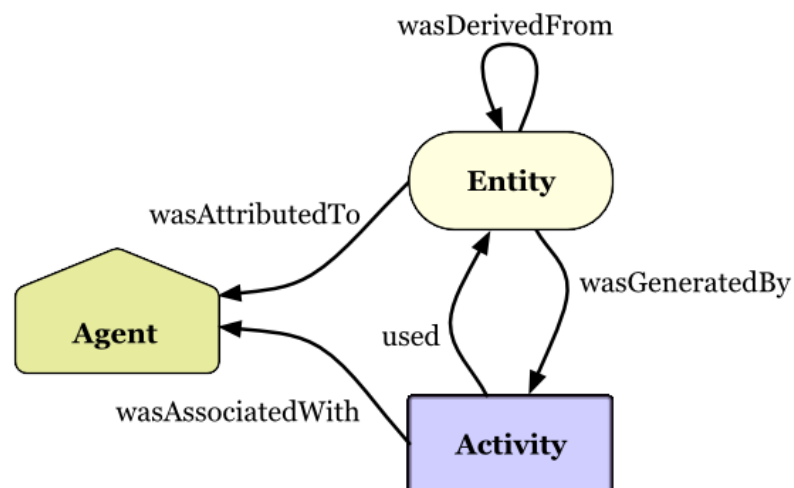


Figure 3.1: Overview model of provenance information stored in the PROV standard.

document (W3C, PROV Model Primer, 2013). Depending on the domain, certain provenance information will be more relevant than others. In GIS, for example, the provenance information is mostly served as metadata describing the data lineage to assess whether the data is useful for the intended analysis. In other cases, provenance information might be more relevant to understand who is responsible for erroneous data or who should be credited. In the case of repeatability, and especially reproducibility, provenance information should be very detailed as the input and output variables of every data transformation should be retraceable. According to Gil et al. (2007), properly recording provenance information is becoming increasingly hard. Collaborations between many scientists in different locations, storing provenance information in highly fragmented systems, including 'emails, Wiki entries, database queries, journal references, codes (including compiler options), and others'. Many different approaches have been suggested to solve these provenance issues, popular is the use of Scientific Workflow Management Systems (SWfMS).

These SWfMS are tools to execute workflows and manage the datasets in different computing environments. As workflows become more and more data-intensive these systems can do parallel processing where possible, and divide the work in the grid or the cloud. Different tasks (or nodes) in the workflow have certain dependencies between each other, and as soon as the dependencies for a specific task have been fulfilled they are started. As the management system starts the different tasks and distributes the input data and intermediate products it is the ideal candidate to track provenance data (Liu, Pacitti, Valduriez, & Mattoso, 2015). On top of creating provenance information based on the tasks that the system performs itself it should also be able to ingest provenance information associated with the input data (Gil, Deelman, et al., 2007) The capabilities of the SWfMSs are often catered to the needs of specific domain. Popular SWfMSs are Pegasus (Deelman et al., 2015) , Taverna (Oinn et al., 2004) for bio-informatics, Galaxy (Goecks, Nekrutenko, & Taylor, 2010) for life sciences and Wings (Gil, Ratnakar, & Deelman, 2007) which includes semantic capabilities. Next to including semantic capabilities in SWfMSs other possibilities to use the Semantic Web to track provenance and deal with workflows have been researched, something that will be discussed in more detail in section 3.5. Independent of the way in which the workflow has been created and the provenance has been tracked, another important aspect of workflow reuse is the way in which it is shared.

### 3.2 Sharing Workflows

Even though workflows have been used for over two decades already there still is a problem when it comes to understanding workflows that have been created by others. Many publications make use of workflows but do not share their workflows in a useful way. In many cases they are merely described in the text but research has shown that the level of detail that is used is not sufficient to recreate the workflow, which leads to insufficient possibilities for reviewers to examine the computational methods, even when the datasets are published (Fang & Casadevall, 2011). Plenty research has been undertaken to find methods to increase the sharing of the actual full workflow rather than describing the workflow in text. The absence of sharing of full workflows is not just reluctance but also the absence of a proper platform to do so. Where for plain code there are collaboration platform such as github, this has long been lacking for scientific workflows. MyExperiment is such an effort to allow easier sharing and finding of workflows. It allows users to upload workflows, with (assisted) metadata creation and a merit system ranking the reliability of workflows (Goble et al., 2010). Another effort focussed more on collaboration between scientist, and which also allows to perform part of the analysis in the workflows online is Crowdlabs (Mates, Santos, Freire, & Silva, 2011). However, even though this allows the sharing of full workflows as they were used during research, workflows are often created with a specific setup of software and code, unique to that specific research. So even though workflows are shared the execution environment has to be copied exactly by other researchers

in order to reuse the workflow, this limits the reusability significantly (Garijo & Gil, 2012). Furthermore, there is little interoperability between SWfMSs, meaning that a workflow created in one system is not necessarily accessible in a different system. Therefore, reusing parts of workflows and combining them can only occur if they are stored in the same format. Efforts to bridge the interoperability issues between different SWfMSs have been undertaken. Oliveira et al. (2016) have developed Géfyra, a architecture based on the PROV-Wf model to translate provenance information from the format of a specific SWfMS to PROV-Wf and from there to other formats. This method proved partially successful, however as PROV-wf does not cover the full capabilities of every SWfMS several operations are lost in the translation process (Oliveira, de Oliveira, & Braganholo, 2015).

Working with provenance translation is an idea that has been proposed more often. Already in 2007, an effort was made to overcome several of these interoperability issues by creating the Open Provenance Model (OPM). OPM has been designed to allow provenance information to be exchanged between systems by means of a compatibility layer based on a shared provenance model (Moreau et al., 2011). This model has successfully bridged some of the compatibility issues between the different WfMS as well part of the documentation issues. However, the model is still focussed on the fully executable workflow that is specific to the designed execution-environment. With this in mind Garijo & Gil (2012) extended OPM with a profile called OPMW. This should allow for the description of abstract workflows which serve as a 'conceptual and execution-independent view of the data analysis method'. However, this method, and the OPM standard both operate from the preconception that the user has an executable workflow to work from and the possibility to work with advanced workflow systems. While their focus lays purely on scientific workflows and creating more abstract version from executable workflows, this research goes a step further. It has been attempted to create a method that not only allows the use of abstract workflows as a way to better understand executable workflows but it should also be possible to only have an abstract workflow as a starting point, as there are still many unknown factors, to create the executable workflow.

When taking the abstract workflow as a starting point the definition of an abstract workflow as a 'conceptual and execution-independent view of the data analysis method' still applies however it is rather a set up towards the data analysis method than an abstraction of it. Taking abstract workflows as a starting point allows users who do not yet have a clear idea of what the executable workflow will look like, to already model and share their ideas. By doing so helping others with similar issues to come to a solution. However, the terms they use and the way they model their ideas should be standardized in some way to allow others to provide a useful and understandable contribution. One way to standardize these terms and workflow elements is the use of the Semantic Web. Something that has also been proposed by Garijo & Gil (2012). They want to use the Semantic Web as a way to share and describe their workflows in order to allow free and open access to a large workflow repository and allowing the workflows to link to other predefined concepts on the web in order to improve their documentation. Before discussing in which way the Semantic Web can assist in standardising workflow description and sharing, a closer look has to be taken at which abstraction methods are available and in what way these abstract workflows can be created.

### 3.3 Levels of abstraction

Abstract vs executable workflow is of course not a clear-cut difference between two possibilities. These are rather two ends of a scale on which any workflow can be placed. Where executable speaks for itself, you push a button and the workflow is executed and the user is presented with the result, an abstract workflow is a more ambiguous term. It refers to a version of a workflow where a lot of factors are unknown. The only known factor is what result should be achieved when using this workflow. However, a workflow in its most abstract form might even have a rather ambiguous result.



When we look at the MaMaSe workflow (Figure 3.2) the intended result for the executable workflow is calculating the total rainfall during the wet season in the MaMaSe area, delivered in a raster format with a 250m resolution and projected in the UTM WGS 84 zone 36S projection. If this would be put into more abstract terms the result of the workflow can be defined as the total rainfall during the wet season in the ideal projection and resolution for the research area. The result in the case of the most abstract workflow as described above would be the aggregation of any natural phenomenon using the optimal resolution and projection for a certain area. However, as in most cases a workflow is designed to solve a specific spatial problem, the most ambiguous result will not be a likely starting point. Therefore, the second level of abstraction described above is most suitable. It is known that the result should be an aggregation of total rainfall in the wet season, however, the workflow in its most abstract sense should also be useable for another area, therefore the projection and resolution should be based on what is most suitable for the selected area and what data is available for that area.

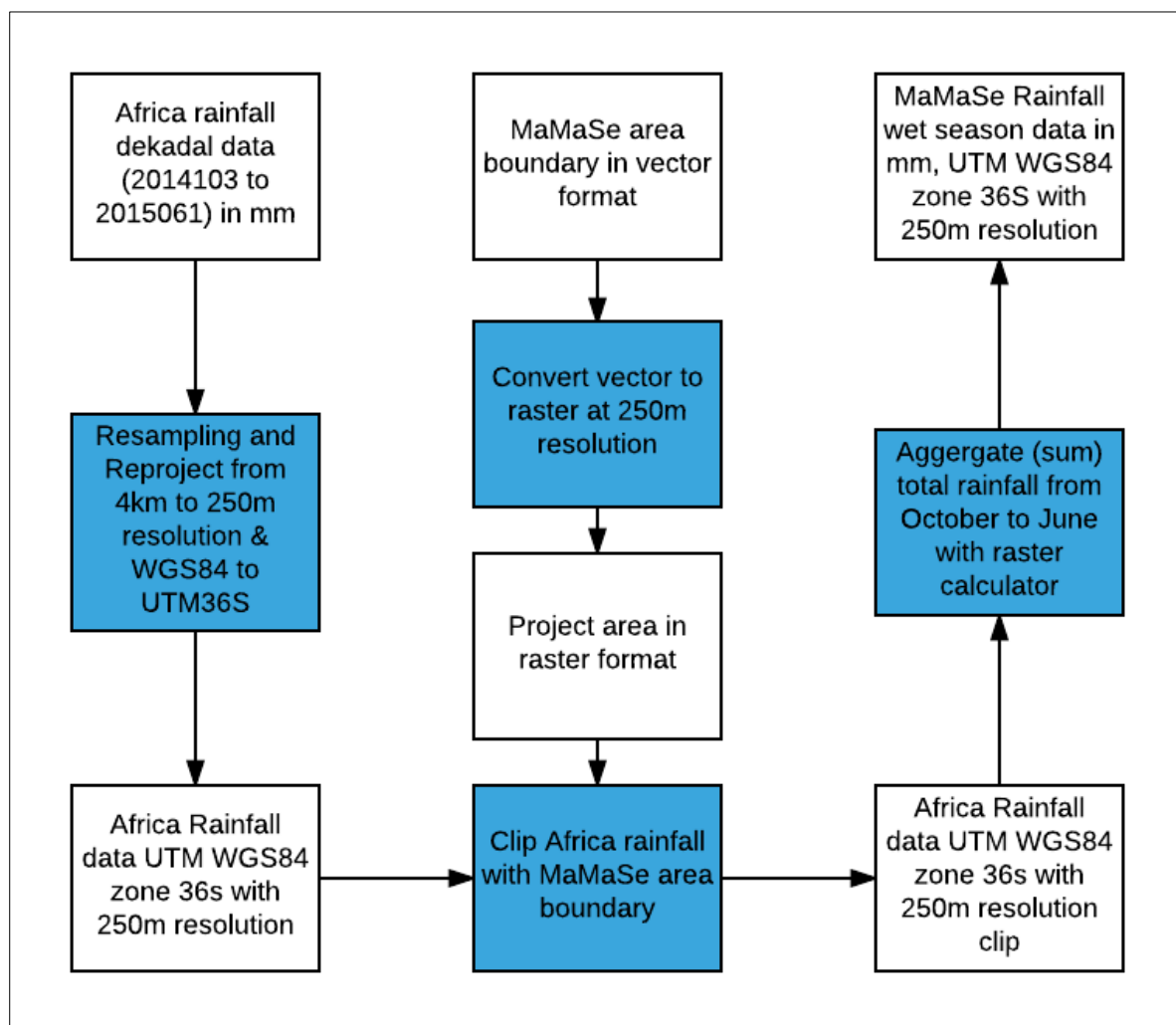


Figure 3.2: Original Use Case Workflow.

### Types of abstraction

The abstraction of workflows is not a new concept, research in this area has already been done and this provides different ideas about the way in which a workflow can be abstracted. Different types of abstraction are suitable for different purposes. In the case of this research it is important that the

abstract workflow has the same number of steps as the original workflow, because this makes it possible to obtain the executable workflow through the specification of each separate step. Two ways of doing this are skeletal planning and step abstraction. In the case of skeletal planning every step in the abstract workflow describes the type of the operation however in a more abstract way, by leaving out specific execution parameter, however it is still possible to trace it back to a specific operation (Garijo, Gil, & Corcho, 2017). In the case of step abstraction, similar operations are classed together providing an overview of the types of steps that should be taken, but it no longer allows for easy translation to an executable workflow. Therefore, less suitable for this specific research, but when taking the abstract workflow as a starting point, it is nevertheless a very useful abstraction method as an intermediate step towards the skeletal plan. Other types of abstraction make use of the clustering of steps to provide a more abstract view. For example, macro abstraction combines several steps and provides more general descriptions of the subsection of a workflow. Finally, layered abstraction provides different levels of abstraction based on the perspective the user is interested in, it doesn't necessarily fit one specific format, but provides the information necessary based for example on the level of expertise of the beholder (Garijo & Gil, 2012).

As mentioned before the level of abstraction can be placed anywhere on a continuum between fully abstract and executable. The workflow in Figure 3.2 is certainly not an executable workflow, it is a diagram describing a skeletal abstraction of a workflow, where it provides some of the inputs such as the output projection and raster size but clearly some are missing such as where to find the data or what the output should be. Furthermore, a list of optional parameters could be defined, such as the resampling method. But considering the level of detail that is there (specific data range, cell size, output projection) it is also far from the most abstract it could be. Especially when taking into account the fact that this is a geo-information workflow, the fact that the output projection and input data have been defined mean that it is not possible to re-use this workflow in a different area without applying further abstractions and changing inputs.

Despite the difficulty of pinpointing the exact location of the workflow on the continuum between abstract and executable it is possible to distinguish some concrete levels of abstraction. Figure 3.3 describes some of the logic that can be applied to the order of these abstraction levels on the continuum when taking into account a

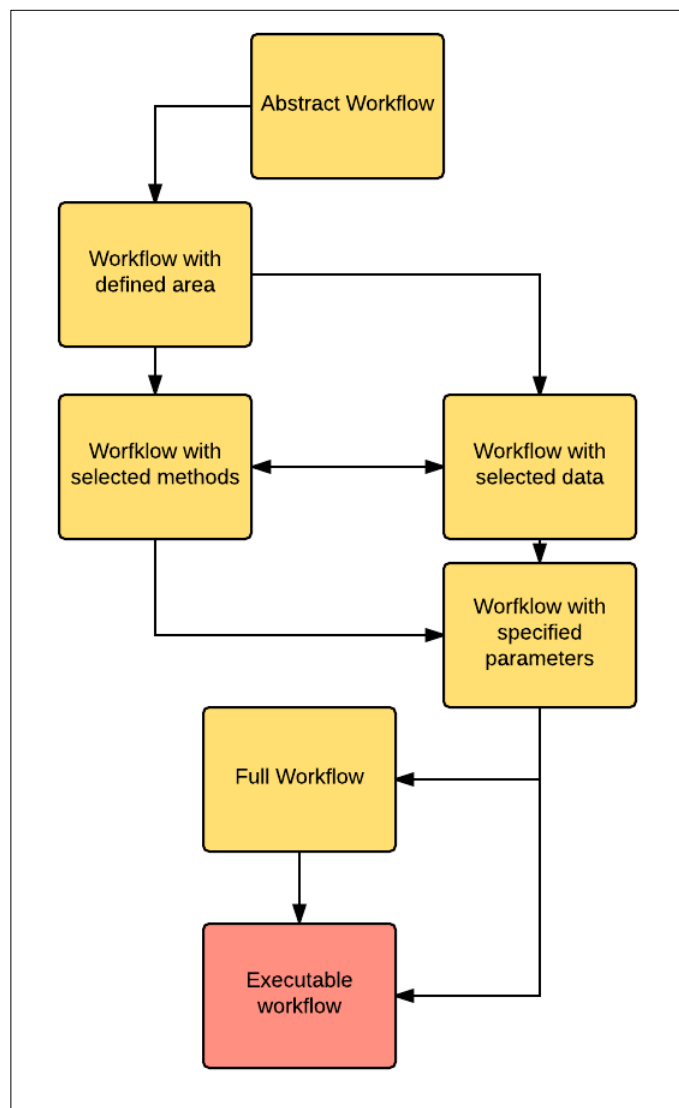


Figure 3.3: Schematic representation of abstraction levels.

geo-information workflow. It takes the abstract workflow as a starting point and considers in which order the specification of an abstract workflow should take place while maintaining the highest level of abstraction possible. This eventually leads to a 'full workflow' which has all the inputs per operations defined but is not necessarily in an executable format, with the executable workflow as a final stage.

**Abstract workflow** – This is highest level of abstraction taken into consideration in this thesis. It is a series of operations in a specified order. Neither the data being used nor the parameters are defined specifically. Considering the MaMaSe workflow in Figure 3.1 the first data field would read 'rainfall data' and the operations are described as 'reprojection' without any specific parameters declared.

**Workflow with defined area** – The next level of abstraction is the same workflow with a defined area. In defining the area, important parameters of the result can be defined. In the case of the MaMaSe workflow this means the optimal projection and resolution can be chosen. This will lead to knowing part of the parameters that are necessary for the operations earlier in the workflow. The most important aspect of the step is that the result can be fully defined. Most of the input necessary for earlier operations can be derived from this, therefore this level of abstraction is scaled higher than the data and parameter selection.

**Workflow with selected data** – At the next level of abstraction also the starting datasets will be known. In many cases the parameters that are selected are dependent on specific features of the selected dataset. For example, in the case of a reprojection it might be necessary to define the projection of the input dataset. So, to create a workflow that defines all parameters, it is necessary to start by knowing the datasets. Furthermore, by knowing the datasets some of the operations in the workflow might no longer be needed, for example, depending on the datatype of the dataset the vector to raster conversion might not be necessary, but a resampling might be.

**Workflow with selected method** – This level of abstraction is where the specific method used for an operation is selected. For example, the resample operations describes different resampling techniques. Based on the data specifics some techniques might be more suitable than others. Therefore, it is logical to place the selection of the methods after the selection of the data, however as multiple datasets might be available, it might be preferred to select a method first, therefore these two levels of abstraction are interchangeable or even at the same level.

**Workflow with parameters** – The next step on the scale towards an executable workflow will be the input of the parameters. Many of the parameters have become forced inputs based on the choices made on earlier levels of abstraction. For example, the input projection is based on the selected data set, and the most suitable output projection is based on the area that has been selected. However, other variables might still be obscured on previous levels of abstraction to leave space for wider applicability of the workflow.

**Full Workflow** – Depending on the format used to describe the levels of abstraction described above, the full workflow might also be an executable workflow. However, this is not necessarily the case, as the above stated levels of abstraction can all be described as diagram or a text document, but also as python code or RDF. Depending on the execution engine used to run the workflow these formats might suffice to make the workflow executable.

**Executable workflow** – In the case the workflow has been created with a specific piece of software, for example the ESRI model builder, then the workflow will already be executable when the parameters have been added. In case the workflow has been described in a different format, making it a full workflow, but not yet executable, some translation should be performed to make it executable.

In conclusion, abstract workflows can be generated in different formats and at different levels of abstraction. However, to perform the step from abstract to executable this research proposes using the Semantic Web to perform this translation. As this allows to add the needed semantics to an abstract workflow, allowing the description of a workflow in a platform and software independent manner, without losing the specifics needed to come to a result intended by the initial workflow.

### 3.4 Semantic Web to the rescue

As already discussed in short before there have been several attempts to use the possibilities of the Semantic Web and Linked Data to solve some of the issues involved in the re-use and sharing of workflows. There are the workflow repositories such as MyExperiment and the ontologies for provenance description such as PROV and OPM. However, these all solve a specific problem of the scientific workflow domain. This while the promise of the Semantic Web is interoperability and the possibility to incorporate different Linked Data applications into one as, for example, an online workflow management system. This next step has been taken in the PhD research of Garijo (2015). He concluded that the provenance models like OPM and PROV only allow for the description of things that have already happened. So the description of an executed workflow (workflow instance) rather than the description of a workflow plan (workflow template).

He proposed the extension of PROV ontology by the P-Plan ontology. This ontology allows the user to describe a *Plan* consisting of several *Steps* which have inputs and outputs defined as a *Variable*. Furthermore, this ontology introduces the *isPrecededBy* property which allows for describing the workflow logic and order in RDF. As the ontology is built upon and connected to the PROV ontology it is possible to create a provenance trace that can be expressed in OPM or PROV. Finally, this is all brought together by a final extension called OPMW which describes workflow templates (abstract workflows) and workflow executions (executed workflows). Furthermore, OPMW provides the possibility to describe attribution, stating who created the workflow template and who was responsible for the specific execution. Figure 3.4 shows how these ontologies are connected. On top of the development and the linking of these ontologies to create a model that can describe all relevant parts of a workflow, scripts were written to automatically convert the execution traces from a SWfMS to a OPMW description of the workflow. Finally, the workflows that are created in this manner and every element of the workflow (all steps, variables, provenance, attribution etc.) are given a URI and are accessible through that URI (in RDF as well as HTML) and can be queried from their SPARQL endpoint, providing automated sharing of the workflows as well (Garijo, 2015). The fact that every element of the workflow is available in RDF and is in a SPARQL endpoint means that it is also possible to search for workflows based on all these elements, where for example repositories such as MyExperiment are limited to the tags and description that is added to the workflow when uploaded. This means that now it is possible, for example, to find all workflows that are attributed to someone working at the University of Mannheim, have been executed at the University of Barcelona, that use rainfall data and include at least one buffer operation.

#### 3.4.1 Workflow ontology and repository

With the work of Garijo in mind, a second look can be taken at the spatial problem-solving application proposed in chapter 1 (Figure 3.5). The model for the spatial problem-solving application can be split

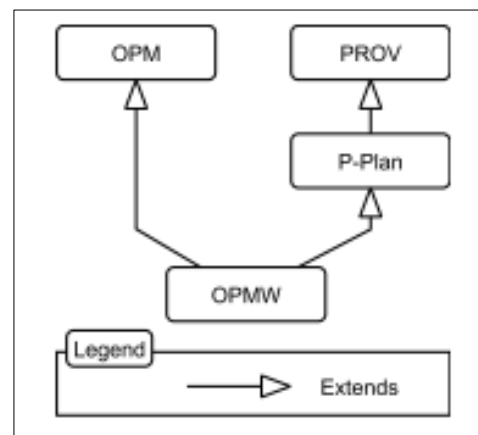


Figure 3.4: Connection between workflow ontologies.

into four parts. Where the first part, the natural language interpretation to find the most suitable workflow for the spatial problem proposed is, albeit a very relevant research topic, too far outside the scope of this research to explore the current state of affairs. However, the other three parts deserve a second look. As explored in the previous sections related work in the field of workflows and the Semantic Web have been very much focussed on the workflow side of the application, step 2 and 3.

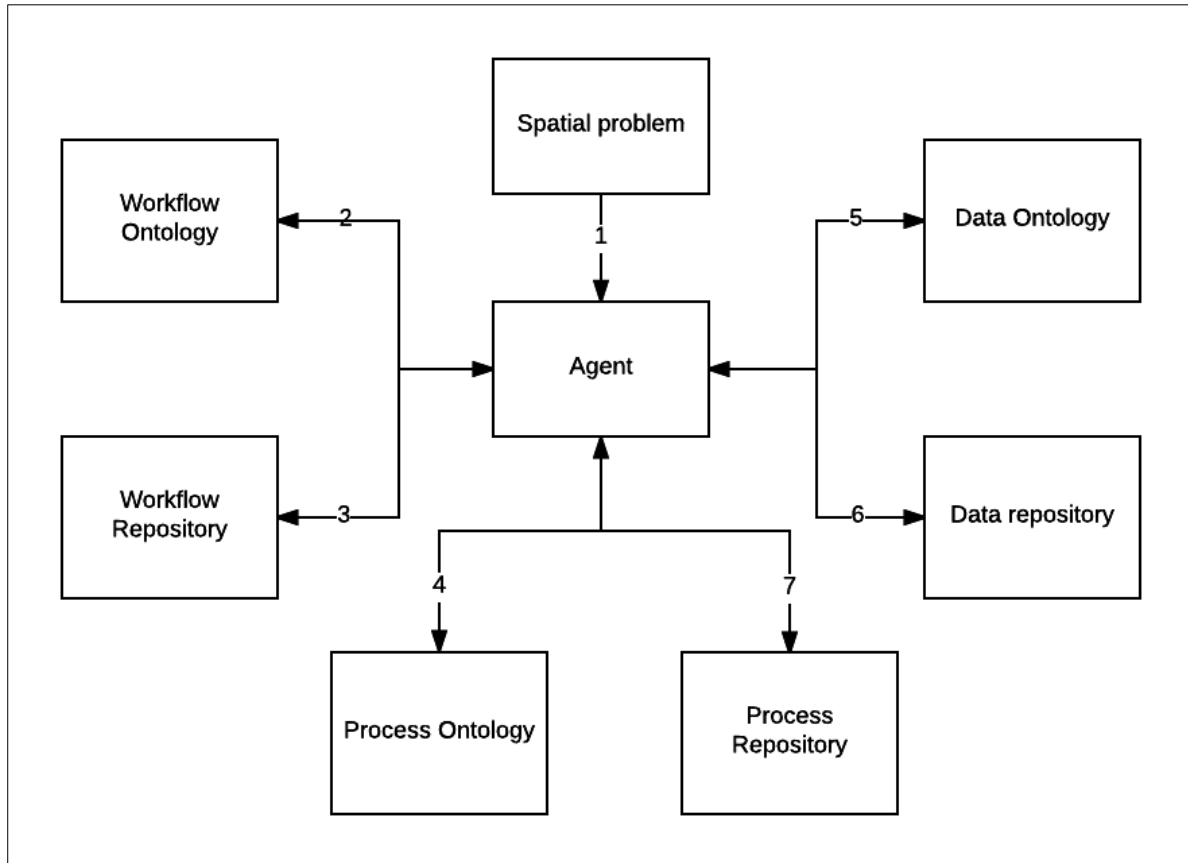


Figure 3.5: Spatial problem-solving application (same as Figure 1.1).

Workflow ontologies have been developed and linked in such a manner that they seem to cover all relevant aspects and through services such as MyExperiment and the OPMW SPARQL endpoint there are also the workflow repositories to share and find these created workflows. If we look at the geo-information side of things there is the workflow ontology created by Diniz (2016). He has developed a workflow ontology which allows the description of GI workflows. However, what is lacking in the work done so far is an ontology or vocabulary which describes specific geo-operations. Where in the OPMW and P-Plan ontologies the focus lies on classes that describe steps and properties that describe workflow logic, it is dependent on field specific ontologies to describe the operations that serve as instances of these classes. In the case of the workflows as described by Diniz, the ontology mainly focusses on describing which input belongs to which operation and what output it creates, but when describing the operations, it refers to Web Processing Services (WPS), for the execution to circumvent the lack of an operation vocabulary. The same goes for the inputs and outputs that are needed in both ontologies. To allow users to create an abstract workflow it is not possible to only apply one of these methods. Instead it will be necessary to be able to refer to exact operations and ideally browse through these operations to be able to select the most appropriate one. These domain-specific operation ontologies are relevant to step 4 and 7 of the spatial problem-solving application introduced

in chapter 1 (Figure 3.5). On top of this there is the need to find the most suitable data relevant to the selected workflow and project area, these are selected in step 5 and 6 of the application.

### 3.4.2 Data ontology and repository

Similar to the standardisation effort in describing workflows and the way they are stored and findable an effort is needed when it comes to the data being used by these workflows. As data comes from different sources and communities, there is need for integrating, translating and validating of data that comes in, to take care of inevitable heterogeneities. Even though the focus of this research is rather in the geo-operation domain than the geo-data domain a short exploration has been undertaken to ascertain to what extent the promise of the Semantic Web as spatial problem solver holds up.

In several domains ontologies describing the way data should be modelled are already available and integrated into the way of working. Especially in the field of bio-informatics where the Gene Ontology (GO) was already available as RDF in 2004 (GeneOntologyConsortium, 2004). Such an ontology, similar to the way OPMW does for workflows describes in which way data should be described, what classes of data are there and what properties should be defined for what type of data. This forces uniformity in data description and allows for easy findability, even automated, when looking for data to use in a workflow. Workflow descriptions in RDF can then also refer to specific types of data that are needed for a certain step, integrating the entire process. In the case of an abstract GI workflow a certain operation might require rainfall data. A geo-data ontology in this case can first provide information on the different types of rainfall data that exist. First of all, the time interval needs to be defined (day, month, year) and secondly the way the rainfall data has been measured (sensor, VGI). When it comes to efforts to provide ways to integrate geo-data into the Semantic Web there are already quite some interesting efforts. A first step was not storing the geo-data as Linked Data but providing ontologies for better discovery of data catalogs. A good example of this is the DCAT ontology. This has been developed in order to 'facilitate interoperability between data catalogs published on the web' (Maali, Erickson & Archer, 2014). This ontology can be used to describe datasets in data catalogs and specify their location. This increases their discoverability and allows applications to find and use them. When it comes to storing the geodata as RDF the most simple ontology that is available is the WGS84 Basic Geo vocabulary which allows the description of points through latitude and longitude properties. A more sophisticated effort came with the GeoSPARQL standard which is based on the Simple Features standard by the OGC, allowing the description of lines and polygons in Well Known Text (WKT) (Iwaniak, Kaczmarek, Strzelecki, Lukowicz, & Jankowski, 2016). Even though this allows for the description of the features as RDF, the properties to describe these features are still, depending on the specific domain, up to the data provider. An effort to standardise and classify types of geodata is still missing.

### 3.4.3 Operation ontology and repository

In line with the developments needed in the geo-data domain, there is a need for the development of ontologies that describe the different classes and structure when it comes to geo-operations. The instances of these classes, describing specific operations, can be used as steps in the OPMW workflow description, integrating the entire workflow process and the selection of possible execution engines into one Semantic Web application. As already discussed by Gil, et al. (2007), creating semantic descriptions of both data formats and required types for a specific operation in a workflow, reasoning and planning capabilities could be added which can add steps to a workflow (e.g. data conversion or transformation steps). Add to this well-defined semantic descriptions of the execution engines and their requirements, automated selection of execution engines could be possible with the relevant dynamic optimization of the workflow. To achieve this, both on the data and operation level there are

efforts needed similar to what has been achieved in the workflow management domain. The previous section already described what efforts have been made in the data field and what should be done. The following chapter will describe the creation of a proof of concept which focusses on step 3, 4 and 7 of the spatial problem-solving application defining a geo-operation ontology and instantiating the operations necessary to transform an example abstract workflow to an executable workflow assisted by the Semantic Web.

## 4 Proof of concept

The aim of this research '*The development of a method to semantically enrich an abstract workflow and convert it to an executable workflow using Semantic Web technologies*' concerns three steps of the proposed spatial problem-solving application. Step three which is concerned with sending an abstract workflow to the application in order to allow semantic enrichment. Step 4 where the application uses an operation ontology and instances of the classes in this ontology to semantically enrich the workflow. Finally, step seven which is concerned with the conversion of the abstract workflow to an executable one. This chapter will first describe the proposed proof of concept and its exact expected capabilities. Furthermore, it will describe the selected example workflow, data and tools being used and explain why these have been selected. After this it the expected results of the ontology and application will be discussed. Finally, this chapter will present the application and ontology in their final form.

### 4.1 The approach

This section will provide an overview of what the proof of concept should achieve, which workflow, tools and data have been selected and why, and finally the expected results of the application and the ontology.

#### 4.1.1 Semantic enrichment and conversion engine

The aim of the proof of concept is to prove that the Semantic Web can be employed to assist a non-expert user in the field of geo-information to convert an abstract workflow to an executable one, using an ontology that describes the possible types of operations and a repository of specific geo-operations that have been instantiated as instances of the classes of this ontology. It should provide enough information to the user to traverse through the levels of abstraction described in the previous chapter and it should output an executable workflow that can be executed in the execution environment preferred by the user. The role the Semantic Web plays in this application is providing all the information in a standardised way, using the same format, allowing it to be seamlessly integrated. Where this specific proof of concept will focus on a couple of specific execution engines, it does allow, by using an openly available standard, to easily add execution engines to the application. This can be done by instantiating operations from other execution engines according to the structure provided by the ontology. This proof of concept simply provides the proof that using one standard format which is platform and software independent can allow people to share workflows in an abstract way without losing the ability to understand it and execute it according to their specific needs. The way in which this is done is the following. The user sends an abstract workflow to the application, the application contacts the operation repository and provides the user with all the necessary information that is needed to understand the different steps of the workflow and subsequently gets enough information to be able to provide the right values for every operation parameter. Finally, the application will then convert the workflow to the appropriate format based on the selected execution environment.

#### 4.1.2 The workflow

Already introduced in the previous chapters the workflow is taken from the MaMaSe project. This has two reasons, first, the MaMaSe project generates workflows of which the reuse benefits are clear. Secondly, the workflows and data are readily available at the supervising institution, the ITC, ensuring minimal distraction from the main goals of this research. Figure 4.1 shows the workflow as it was available at the ITC. The workflow is suited for this specific research as it consists of operations that are available in most GIS, which allows the proof of concept to be tested using multiple execution engines. Furthermore, it uses both raster and vector data which can test the working of the application with both types of data. Finally, the workflow is of an appropriate size to consist of enough operations



to see the versatility of the application without being too extensive to be too time consuming to convert and execute during testing of the application. The workflow consists of rainfall data as input which is converted to the right projection and raster size for the MaMaSe project area. Subsequently a clip operation, using the rainfall data and the boundary of the project area, which has been converted to raster from its original vector format, to obtain the right subset of the rainfall data. Finally, the rainfall data, which consists of a temporal range of rasters spanning the entire wet season in the project area, will be aggregated using a raster calculation to find the rainfall for the entire wet season in the MaMase project area.

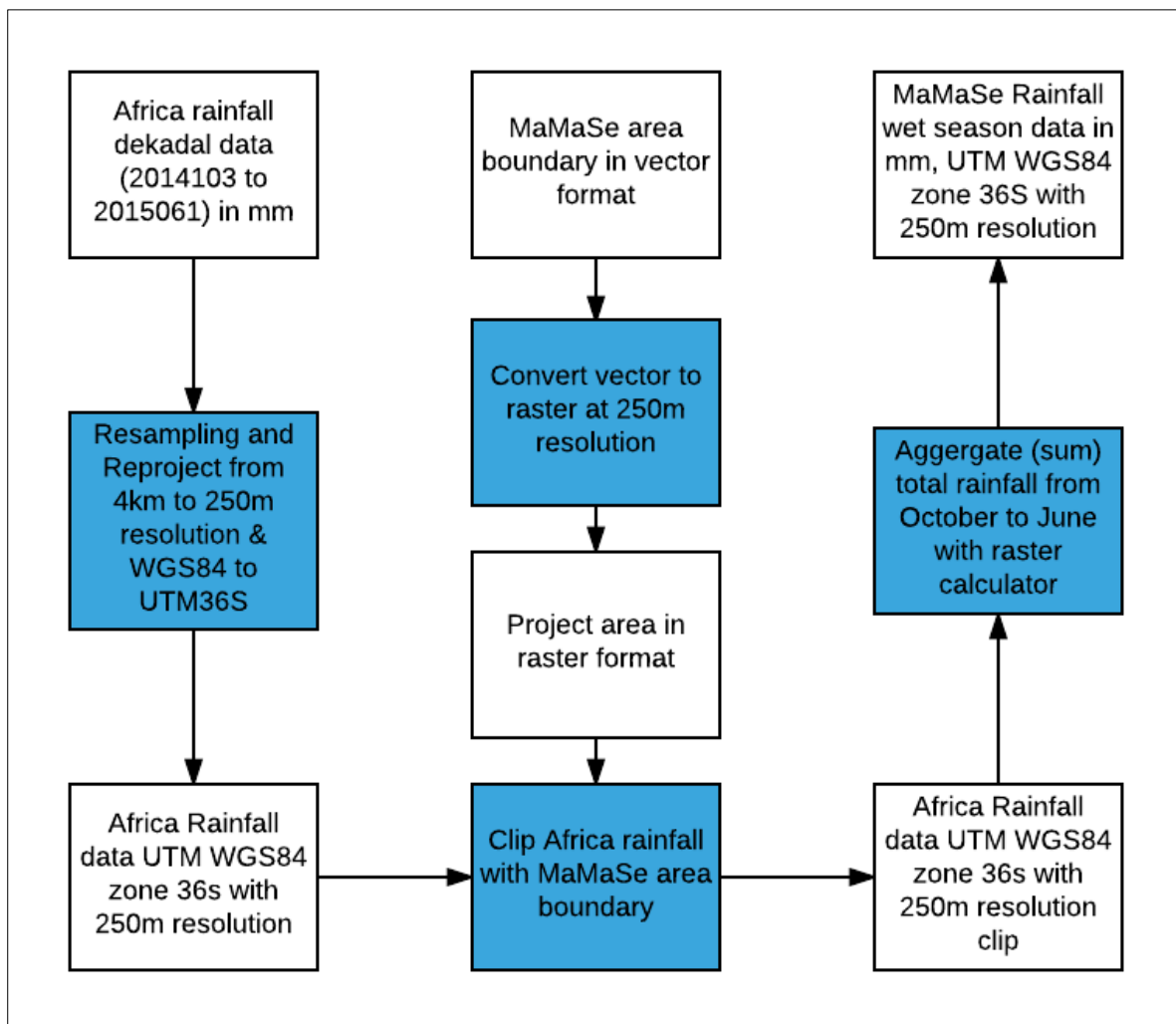


Figure 4.1: Original Use Case Workflow.

Figure 4.2 shows the same workflow but in the most abstract form as it will be used as an input in the proof of concept. To be used as an input the abstract workflow needs to be recreated in a format that is machine readable, which has been done using the ILWIS model builder, which provides a JSON output of the workflow, this process will be discussed further on.

#### 4.1.3 The data

The data used in this proof of concept is twofold. First, there is the RDF data that has been developed as part of this research. This data will be discussed later in this chapter, in section 4.2. The other data involved is the geographic data, consisting of rainfall data and the boundary of the project area. Both datasets have been provided by the ITC as the original data used in this workflow. The boundary data is a vector dataset containing only the boundary of the MaMaSe project area, which covers part of

Tanzania and Kenya. The rainfall data is a set of rasters describing total rainfall per 10 days in mm from the end of October 2014 till the start of June 2015. This data has been collected by GEONETCAST, and extracted through the GEONETCAST extension of ILWIS. It is in-situ station data that incorporates satellite imagery in a 0.05° resolution. The dataset covers the entire African continent.

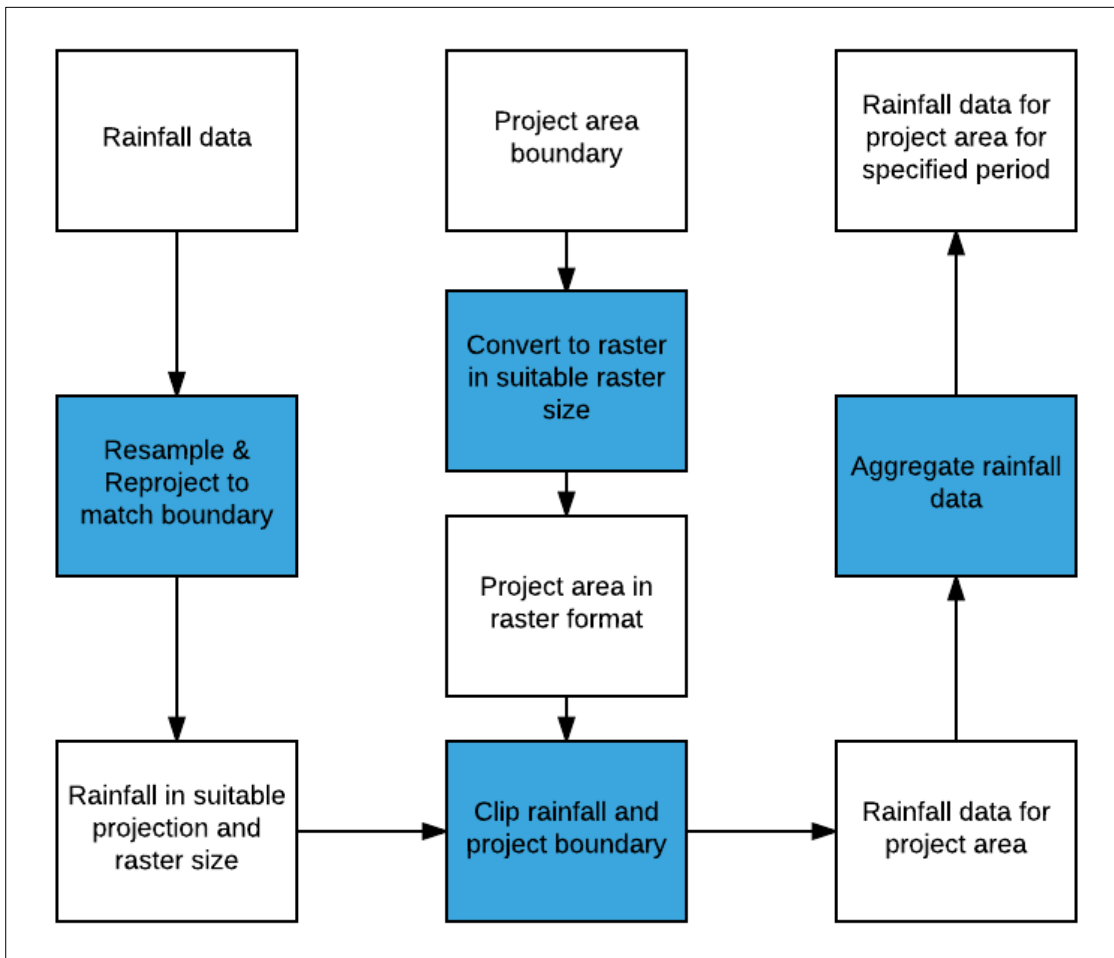


Figure 4.2: Abstract version of Use Case Workflow.

#### 4.1.4 The tools

During the creation of the proof of concept and the data preparation several tools have been used. There are a few criteria that have been used for the overall tool selection. Being that it is a study into the Semantic Web which promotes the use of open data and advanced sharing of data, it is important that the tools being used are, as much as possible, freeware or open source software. This allows for anyone to make use of the developed method. Furthermore, wherever it is the case a choice has been made to make use of existing, broadly accepted, standards and multi-platform tools. This means that it can easily be used by anyone, regardless of their current setup. This section will describe them per type of application.

#### Editors

Two editors have been used during the creation of the proof of concept.

*RDF-editor* - For the RDF documents RDF Editor has been used. This has been selected because of its simplicity and ability to quickly translate RDF documentation from one serialisation to another. Allowing translation to turtle to make the RDF documents more human readable.

*Notepad ++* - For non-rdf documents, for example XML or JSON, notepad ++ has been used. This is a simple text editor, however it supports syntax checking and offers support for many languages, which allows for better readability. Furthermore, this defeats the need for using a new editor for every new language that is encountered.

### **Execution environments**

For the execution of the executable workflow a selection of two GIS has been made. This has been done because the expected software independent solution that is provided needs to be tested by being able to execute the workflow in two different execution engines. On purpose, two different types of GIS have been selected. One open source and one proprietary. Both have built-in python capabilities, which is an ideal format for the executable workflow, in that it is a platform independent language and the workflow does not have to be translated to a software specific format.

*ILWIS* – The Integrated Land and Water Information System (ILWIS) is an open source GIS developed by the ITC. The selection of this GIS is based on two reasons. They recently developed a new version called ILWIS Objects which allows the use of python script to call certain functions and execute the entire workflow from the command-line. Allowing execution by non-expert users by simply running the script that is created by the application. Furthermore, the new model builder supports exporting the workflow as JSON. This means that this software can be used to create the abstract workflow in a format supported by many programming languages, so it provides broad usability, whichever language would be used to develop the proof of concept.

*ArcMap* – Probably one of the most well-known GIS, developed by ESRI, is ArcMap. The selection of ArcMap as second GIS is also twofold. Primarily the reason is the integrated python library ArcPy, which like ILWIS Objects allows for easy creation of the executable workflow. Furthermore, as it is one of the most used GIS, integrating this into the application would allow for most widespread use of the application.

### **Development**

Several development environments were needed for the creation of the proof of concept. One for the building of the application, one for the development of the ontology and one for the testing of the python code for the executable workflow. For the creation of the application JavaScript has been used, as this is the standard language for web-applications.

*NetBeans IDE* - NetBeans is a cross-platform open source integrated development environment (IDE) which among other languages supports JavaScript. It relies heavily on the developer community and allows extension by third party modules. Furthermore, it automatically updates the dependencies for every project after a supporting module is installed. This means that when the project is exported to a different user, the necessary extra modules are automatically installed on the seconds user's system. These functions combined with the open source nature of the software make it the ideal IDE to use for this project.

*Protégé* – As explained in chapter two, protégé is an ontology builder. It uses the Web Ontology Language (OWL) and has an easy to use GUI and more importantly and build-in reasoner that helps in checking the correctness of the ontology that has been developed. There is currently no open source software that provides the same level of functionality, so this tool has been used to create the ontology.

*IDLE* – IDLE is the most common python IDE as it has been bundled with the default implementation of the python language since version 1.5. It provides basic functionality and is completely cross-platform and open source. Meaning anybody can use it. As the level of python development in this proof of concept is limited, the standard python IDE suffices and has been used for the development of code for the executable workflow.

### Triple store

As described in chapter two, a triple store is the database of the Semantic Web. Both the ontology and the operation repository are available through the triple store. Several triple store have been discussed in section 2.5.1. Initially, Parliament seemed the right choice because of its full implementation of the GeoSPARQL implementation. However, as in the proof of concept the geodata is not stored as RDF, GeoSPARQL has not been used. Furthermore, Parliament had some issues with the version of Java installed for some other tools, therefore the switch has been made to GraphDB. GraphDB is also a free to use triple store, with a more sophisticated GUI, and no noticeable performance differences with the number of triples stored for this proof of concept.

### 4.2 The ontology

For the proof of concept application to support the user in converting a workflow from abstract to executable the way the geo-operations are described need some level standardisation. For this purpose, an ontology has been developed. Such an ontology describes classes of things and their

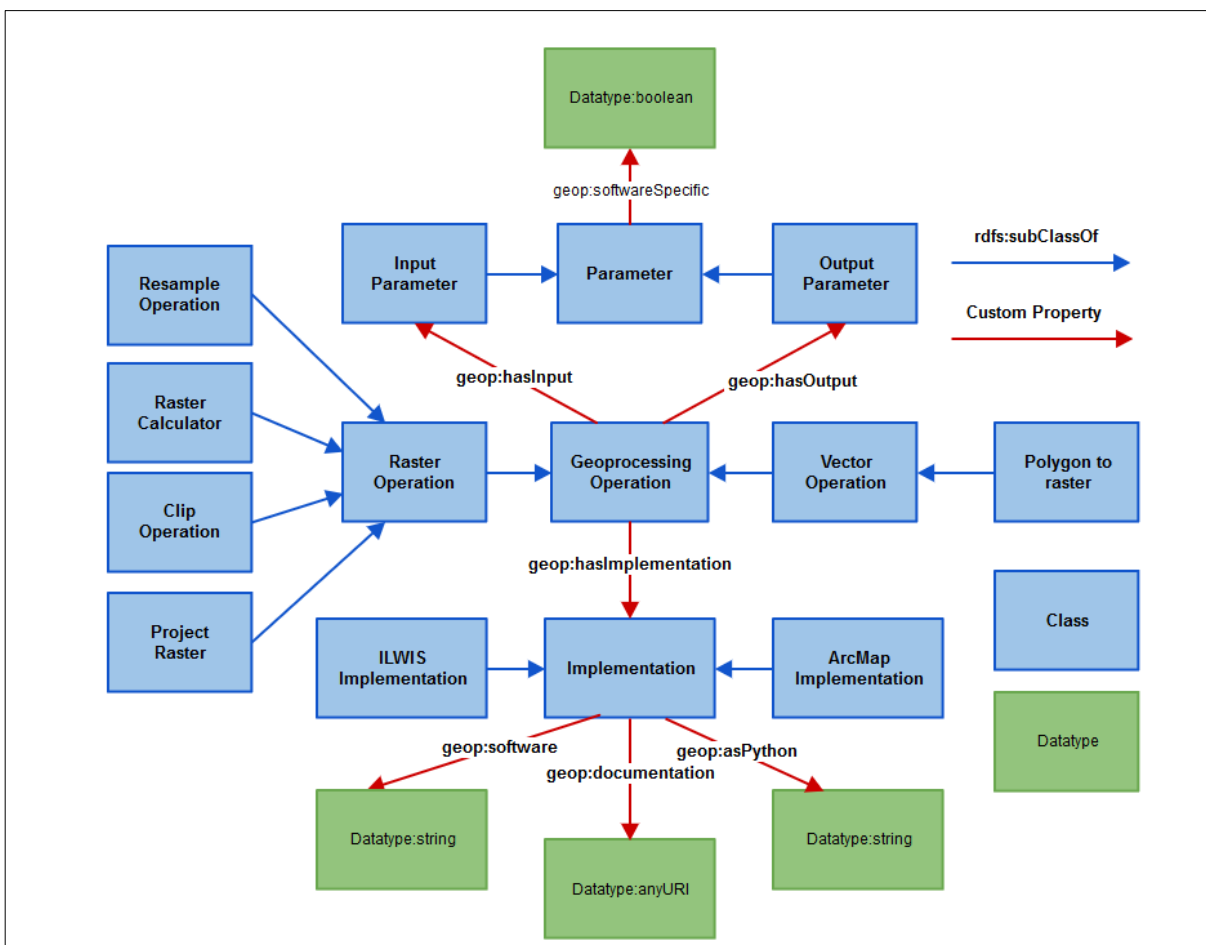


Figure 4.3: Schematic overview of the GeoOperation Ontology.

relationships and provides some structure. To develop an ontology that encompasses all geo-operations their relations and possible implementations in different GIS is a task that is far beyond the scope of this research. The ontology developed here should be seen as a set-up, or suggested structure, that could be used as a basis for developing such an all-round ontology. It contains a basic structure and provides enough detail to describe all operations in the proof of concept and their implementations in both ArcMap and ILWIS.

The ontology consists of 14 classes and 9 properties, a schematic overview of these can be found in Figure 4.3. This shows the central class being the *Geoprocessing Operation*, every operation is a member of this class. This has been subdivided into two subclasses; *Vector operation* and *Raster Operation*. These two subclasses have further subclasses describing different types of operations. All subclass relations have been modelled using the `rdfs:subClassOf` predicate. The *Geoprocessing Operation* has three object properties, properties that define a relationship between individuals of different classes, that have been created for this ontology in the *GeoOperation Ontology*, using the prefix *geop*. For every geo-operation, these properties can be used to define inputs, outputs and point to the specific implementation of an operation. The classes *Input Parameter* and *Output Parameter* are both subclasses of the *Parameter* class. The parameter class has one data property, a property that relates to a literal or something with an XML schema datatype, called `geop:softwareSpecific`. This property allows values of the datatype Boolean, so either true or false. This is needed because in both ILWIS and ArcMap there are certain parameters that are only used in their specific implementation. By allowing this to be described, it provides a distinction between parameters that are valid for all individuals in this class (e.g. input raster, output raster) and parameters that are specific to the implementation (e.g. georeference as specific requirement for the ‘project raster’ operation in ILWIS). By defining this as a property of the class parameter, both subclasses, *Input Parameter* and *Output Parameter*, inherit this property.

For the actual instantiation of the operations per execution engine the two subclasses of *Implementation* have been created; *ILWIS Implementation* and *ArcMap Implementation*. These classes both have three data properties associated with them; `geop:documentation` (datatype:anyURI), `geop:asPython` (datatype:string) and `geop:software` (datatype:string). The `geop:documentation` property refers to an URL where the official documentation for that specific operation can be found. The `geop:software` property refers to a literal where the name of the GIS can be specified. Allowing the user to find all operations for a specific execution engine by searching for its name, for example ‘ArcMap’. This distinction is further required to provide the user with the right documentation and parameters by using it as a filter in the SPARQL query. The final property `geop:asPython`, refers to a code snippet presented as a string, which provides the python code needed to execute that specific operation in the related execution engine. Again, all these properties are related to the Implementation class as then both subclasses automatically inherit these properties. Finally, all classes and properties have two standard annotation properties, `rdfs:label` which provides a human readable label instead of the URI when the operation is presented in the application, and

```
geop:poltoras_am a geop:ArcMapImplementation,
    geop:PolygonToRaster,
    owl:NamedIndividual;
geop:asPython """arcpy.PolygonToRaster_conversion(PolygonToRasterInputFeatures,PolygonToRasterValueField,
    PolygonToRasterOutputRaster, '', '', int(PolygonToRasterCellSize))
    print \"Polygon to Raster succesful\""""^^xsd:string;
geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/conversion-toolbox/polygon-to-raster.htm"^^xsd:anyURI;
geop:hasInput geop:cellSize,
    geop:inputFeatures,
    geop:valueField;
geop:hasOutput geop:outputRaster;
geop:software "ArcMap"^^xsd:string .
```

Listing 4.1: RDF instantiation of Polygon to Raster operation.

*rdfs:comment* which provides a description of what the class contains and what relationship the properties describe. All the operations that occur in the workflow have been instantiated according to the GeoOperation ontology as data preparation step for the proof of concept. Listing 4.1 shows an example of this; the instantiation of the polygon to raster operation in the ArcMapImplementation class.

### 4.3 The Application<sup>1</sup>

As the ontology and its instances have been added to the triple store, this section will look at the application that will make use of this data. This section will first discuss how the input, the abstract workflow, has been created. After this a step by step coverage of the functionality and logic of the application will be provided and design choices will be highlighted.

#### 4.3.1. The abstract workflow

As discussed before, the abstract workflow has been created with the ILWIS model builder. The reason for this is that the output is JSON, which is a format easily usable in a JavaScript application. The way the workflow is created by ILWIS is aimed at creating an executable workflow, therefore some adjustments have been made to make it more suitable, while keeping the workflow logic the way that it was developed in ILWIS. Part off the initial output of the ILWIS model builder when attempting to create an abstract workflow can be seen in Listing 4.2. It shows the first operation of the workflow and its first input. The operation has an id and metadata attached to it. This metadata is quite extensive however, either irrelevant to an abstract workflow, or irrelevant as it is ILWIS specific and

```
"operations": [
  {
    "id": 0,
    "metadata": {
      "longName": "ProjectRaster",
      "description": " ",
      "syntax": "",
      "resource": "",
      "keywords": "",
      "inputParameterCount": 3,
      "outputParameterCount": 1,
      "final": false
    },
    "inputs": [
      {
        "id": 0,
        "url": "veg.com",
        "term": "",
        "type": "map",
        "value": "",
        "units": "",
        "min": "",
        "max": "",
        "name": "InputRaster",
        "show": true,
        "change": true,
        "description": "",
        "picture": ""
      },
      {
        "id": 1,
        "name": "InterpolationMethod"
      }
    ],
    "outputs": [
      {
        "id": 0,
        "name": "OutputRaster"
      }
    ]
  }
]
```

Listing 4.2: Original output ILWIS model builder vs edited output.

<sup>1</sup> For a description of how to install and test the application see Appendix 5

the proof of concept workflow should be software independent. Furthermore, per operation an array of inputs is described, also with a list of values, of which for this purpose only the id and name property are relevant. That is why for the final abstract workflow used for this application all irrelevant information has been removed. The operation description on the right in Listing 4.2 show the first operation of the abstract workflow as it has been used as input for the proof of concept. For the full abstract workflow see appendix 2.

```

"connections": [{
  "fromOperationId": 0,
  "fromOutputId": 0,
  "toOperationId": 2,
  "toParameterId": 0
},
{
  "fromOperationId": 1,
  "fromOutputId": 0,
  "toOperationId": 2,
  "toParameterId": 1
},
{
  "fromOperationId": 2,
  "fromOutputId": 0,
  "toOperationId": 3,
  "toParameterId": 0
}
]

```

Listing 4.3: Workflow logic as created by ILWIS model builder.

In the case of this application all information on the other aspects such as descriptions of operations and parameters will be taken from the triple store, so do not have to be described in the workflow. However, even though the parameters are also defined in the RDF representations, to fully capture the workflow logic, it is necessary for the parameters to be described in the input workflow as well. This workflow logic will later be needed for the application to assist the user in the conversion process. This workflow logic is added to the abstract workflow after the operations and parameters have been defined, as can be seen in Listing 4.3. Defined are the connections between the operations and more importantly which output should serve as which input for the next operation. This is necessary as in workflows multiple outputs may be used as input for a certain operation. The user should not be asked to research the workflow to determine this. This logic should be provided by the application.

For example, in this workflow the output of the project raster operation (1) and the output of the polygon to raster operation (2), both serve as input for the clip operation (3). Where it is quite important which output is used as which input to obtain the right outcome of the workflow.

#### 4.3.2 Semantic Enrichment

The application has been developed using Node JS. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It runs server side and handles request from the browser. It is designed to build scalable network applications, and it is asynchronous and event driven. This means that it can handle many connections concurrently, for every request that is made to node a specified function is executed, but when no connections are made node is sleeping. This tool is used because it is based on JavaScript which is the most suitable and often used language for web applications. Furthermore, it allows very easy inclusion of modules developed by other developers which means that already developed methods can easily be reused instead of coding everything from scratch. The modules that have been used are the following:

- *sparql* - a simple, low-level SPARQL client. This module allows you to define a SPARQL client by defining the sparql-endpoint that should be used. This client has a function which takes a sparql-query as an argument and provides the query results as a JSON-file.
- *body-parser* – is a middleware that parses the body of incoming requests. This is necessary to be able to perform operations on the body of the requests.
- *express* – small, robust tooling for HTTP servers. This module creates the possibility to create a local server with just a couple of lines of code.

- *multer* – middleware for handling multipart/form-data. This module is primarily used for uploading files and allows the use of a html-form to upload the JSON-workflow to the server.

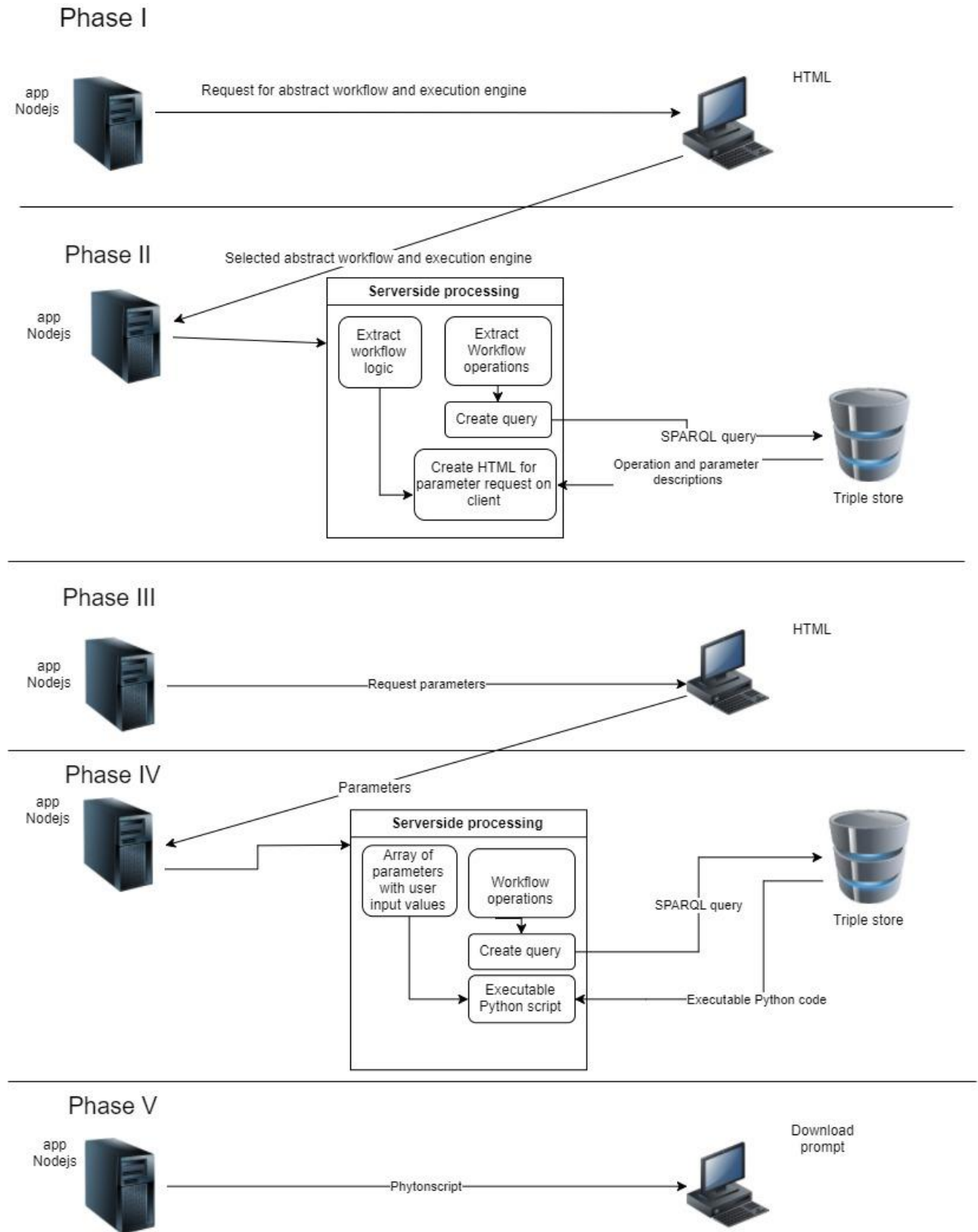


Figure 4.4: Functional model of the proof of concept application



Figure 4.4 shows a functional model of the way the application and all its parts work. It consists of communication between three elements. The application developed in Node.js where all server-side processing occurs, the triple store where the ontology and operation instances are stored and the client where the application is served to the user. The use of the entire application consists of five phases. In the first phase the user is served an HTML page, where he selects the abstract workflow for conversion. The second phase is triggered, where on the server-side operation descriptions are queried from the triple store and combined with workflow logic extracted from the workflow. A new HTML page is dynamically created based on these inputs and in phase three this is served to the user. Here the user provides parameter inputs, which are sent back to the server, for phase four. In phase four, executable python code is queried from the triple store, and combined with the user inputs for every parameter an executable python script is created. In phase 5, this executable workflow is served to the user, who receives a download prompt and can save the workflow to disk. The following section will provide a more in-depth discussion of the application.

### Phase I and II

Using the express module, the first step for the application is the creation of a server on which the application will run. The module also exposes a folder which contains html files that need to be served to the user. This folder can also be used to store intermediate data or other files. The first page presented to the user is a simple html page, allowing the user to specify the abstract workflow that should be semantically enriched (Listing 4.3). There is a check on the type of workflow uploaded, must be JSON, but there is no check possible on the structure of the workflow, this has to be as specified in

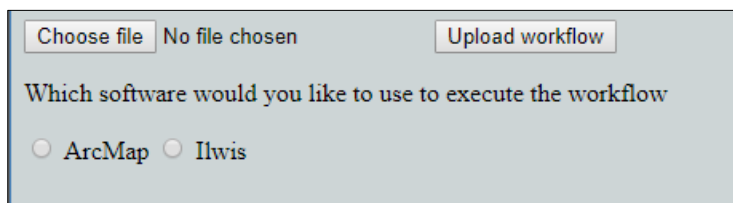


Figure 4.5: Workflow upload and software selection screen.

the previous section. Furthermore, in this screen the user can select which execution engine will be used for the executable workflow. Based on this the application will create the appropriate query and request the user input specific to the execution engine. When the workflow is

uploaded by the user, the application does two things. It extracts all the operation names and adds them to an array, which keeps the execution order of the operations. Secondly, it extracts the parameter relations, as specified in the workflow. This logic is stored in a second array in key value pairs, in unique names generated from the operation and the parameter (e.g. ProjectRasterOutputRaster-ClipOperationInputRaster). This naming convention has been used throughout the application to ensure unique names and optimal cooperation between different parts of the application. The application consists of three important functions that have been developed. The first is the query builder (Listing 4.4). This function uses a for loop to iterate over the operations

```
for (k = 0; k < operations.length; k++) {
  if (k < (numberOfOperations - 1)) {
    query += ' select ?label ?documentation ?input ?output ?inputlabel ?outputlabel ?inputcomment ?outputcomment \n\
    where { \n\
    <http://www.semanticweb.org/ontology/GeoOperations#' + operations[k] + '> \n\
    <http://www.w3.org/2000/01/rdf-schema#label' + operations[k] + '> ?label . \n\
    ?operation a <http://www.semanticweb.org/ontology/GeoOperations#' + operations[k] + '>; \n\
    <http://www.semanticweb.org/ontology/GeoOperations#software' + software + '>; \n\
    <http://www.semanticweb.org/ontology/GeoOperations#documentation' + documentation + '>; \n\
    <http://www.semanticweb.org/ontology/GeoOperations#hasInput' + input + '>; \n\
    <http://www.semanticweb.org/ontology/GeoOperations#hasOutput' + output + '>; \n\
    ?input <http://www.w3.org/2000/01/rdf-schema#label' + inputlabel + '>; \n\
    <http://www.w3.org/2000/01/rdf-schema#comment' + inputcomment + '>; \n\
    ?output <http://www.w3.org/2000/01/rdf-schema#label' + outputlabel + '>; \n\
    <http://www.w3.org/2000/01/rdf-schema#comment' + outputcomment + '>; \n\
  }
  }
}
```

Listing 4.4: Dynamic query builder.

```
function build(json) {
  data = json.results.bindings;
  res.render('index.jade', { test:data, connection:JSON.stringify(connections)});
}
```

Listing 4.5: Build function to dynamically create html page.

array and for every operation it adds a section to the query. Next to dynamically adding the operation names to the query, it also uses the specified execution engine, to obtain the relevant results. This SPARQL query asks for the operation label, input labels, output labels, description of the required inputs, and links to the official documentation. These will be presented to the user in the next screen of the application. The second function it uses is a build function (Listing 4.5) which uses the result set of the SPARQL query to dynamically build an html page, using the Jade templating engine. This allows the use of loops and conditionals to build html based on input data. This function sends the data, and information on the connections between the operations and parameters to the index.jade file. These functions are initiated by the query function, defined in a separate sparql-module (Listing 4.6). This sparql module provides a query function, specifies the SPARQL endpoint which needs to be accessed and then calls the build function with the results of the query as input.

```
function query(query, build) {
  var client = new sparql.Client('http://localhost:7200/repositories/Thesis2')
  client.query(query, function(err, res) {
    if (err) throw err;
    console.log('Query was succesful');
    build(res);
  });
}
```

Listing 4.6: Query function to acquire operation data.

### Phase III

After the build functions generates the html the user is taken to the second screen, where a list of operations is presented. For every operation there is a link to the documentation as it is provided by the developer of the GIS (Figure 4.6). This documentation provides information on how the operations are executed in the software, and what types of values are valid for the different parameters. Next to operations and documentation there is a list of parameters per operation, and input fields where the user can specify the values that should be used for every parameter (Figure 4.7). This part of the application uses the connections between the operations specified in the workflow. As soon as the

Operation	Documentation
Project Raster	<a href="http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/project-raster.htm">http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/project-raster.htm</a>
Polygon To Raster	<a href="http://desktop.arcgis.com/en/arcmap/latest/tools/conversion-toolbox/polygon-to-raster.htm">http://desktop.arcgis.com/en/arcmap/latest/tools/conversion-toolbox/polygon-to-raster.htm</a>
Clip Operation	<a href="http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/clip.htm">http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-toolbox/clip.htm</a>
Raster Calculation	<a href="http://desktop.arcgis.com/en/arcmap/latest/tools/spatial-analyst-toolbox/raster-calculator.htm">http://desktop.arcgis.com/en/arcmap/latest/tools/spatial-analyst-toolbox/raster-calculator.htm</a>

Figure 4.6: Screenshot of the proof of concept application providing the user with overview of workflow operations and documentation.

Input	Inputfield	Info	Output	Outputfield	Info
Cell Size	1000		Output Raster	projOut	
Input Raster	chirps				
Input Georeference	4326				
Output Coordinate System	32736				
Cell Size	1000		Output Raster	outputBound	
Input Features	boundaries_mamase.shp				
Value Field	FID				
Input Raster	projOut		Output Raster	outputClip	
Rectangle	outputBound				
Expression	outputClip		Output Raster	rainRaster	

Figure 4.7: Screenshot of the proof of concept application showing the user input fields.

user selects a name for the output of an operation, this same name is automatically copied in the right input field. This makes sure that there is no possibility for spelling errors, and the operations are connected in the correct way. To guide the user in specifying the correct user input, in the correct way, each input and output field is accompanied by an 'Info' button. When this button is clicked a popup appears explaining what kind of input and in what format is needed. For example, for the Output Raster field the Info button provides the description 'Specify the name of the raster you are creating, no extension needed'. This helps the user, and takes away the burden of going through the official documentation. For a full overview of parameter descriptions see Table 5.1. After the values for every parameter have been specified by the user, and the submit button is clicked, these values are send back to the server to start the process of converting the workflow to executable in the right format.

Parameter	Description
Cell Size	Specify the cell size of the new raster
Input Raster	Specify the input raster for this dataset, unless a value is already present in this input field. If this is the first input raster of the workflow specify a unique section of the filename without extension. (e.g. Filename: AfricaRainfall012016.asc, input: Africa).
Rectangle	Input a raster dataset of the extent that needs to be clipped from the first input dataset
Second Input Raster	Second input raster for comparison operation with first raster
Expression	Specify the input raster for the raster calculation
Input Feature	Specify a vector dataset including extension
Input Georeference	Specify the geographic projection of the input dataset by EPSG code. See (see: <a href="http://spatialreference.org/ref/epsg/">http://spatialreference.org/ref/epsg/</a> )
Output Coordinate System	Input the EPSG code for the coordinate system the dataset needs to be projected to (see: <a href="http://spatialreference.org/ref/epsg/">http://spatialreference.org/ref/epsg/</a> )
Value Field	The field in the attributes table used to assign values to the output raster
Resampling Method	The method used to aggregate the geometry of the output map, see documentation to make a choice
Georeference	Specify the file name including extension of a raster with the preferred projection and cell size, or a georeference file containing this information.
Output Raster	Specify the name of the raster you are creating, no extension needed

Table 4.1: Overview of parameters and their description.

### 4.3.3 Conversion to executable (Phase IV and V)

The conversion part of the application makes use of the earlier described naming convention to match the user inputs to the right part of the python script that is created. The input fields that have been used to specify the parameter values all use this naming convention which means the data that is send back to the server is an array of key – value pairs specifying the parameter name and the user input (e.g. ProjectRasterOutputRaster – ‘rainfallUTM36s’). The conversion operation uses these names to start the python script with declaring all these variables followed by the input values (Listing 4.7).

```
for (var x in pythonInput) {  
    variable = x + ' = "' + pythonInput[x] + '"\n';  
    fs.appendFile('pythonWorkflow.py', variable, function (err) {  
        if (err) throw err;  
    });  
};
```

*Listing 4.7: Function creating Python variables from user input.*

Depending on which execution engine has been selected a specific start to the script is added after the variable declaration (Listing 4.8). This contains the import of specific libraries that are required for the execution (e.g. arcpy) and the declaration of the working directory, which is taken from user input.

```
arcmapstart = 'import arcpy \n\  
from arcpy.sa import * \n\  
arcpy.CheckOutExtension("Spatial") \n\  
arcpy.env.workspace = workingDirectory \n\  
';
```

*Listing 4.8: Code snippet to set up executable ArcMap workflow.*

Then, in a similar fashion to the way the operations and parameters were queried from the triple store, again the operations array together with the selected execution engine is used to dynamically create a SPARQL query to find the python code needed to execute the operations. The prepared pieces of code use variables that have been created, again using the same naming convention. The earlier described query function is called again, which now sends the result to the workflow builder function, which appends the pieces of code to the python script in which the variables have been previously declared. As the names match, the user specified values are automatically used in the right operations. For the full executable workflows see appendix 3 & 4. When the workflow has been created, the user is presented with a download prompt, asking for a location to save the executable workflow. After which the user can easily execute the python script to obtain the requested workflow output.

## 5 Analysis of results

This chapter will look at the process of creating the proof of concept that was presented in the previous chapter. Where the previous chapter presented the results as they turned out, it omitted the analysis of these results and the process of getting there. This chapter will serve as an evaluation of the proof of concept, discuss the benefits of the final result, but more importantly the limitations and provide insight into how these limitations might be resolved. The analysis will be provided in five sections. The first section will provide some insight into the limitations of the testing of the proof of concept which provides some context in which light these results should be reviewed. The second section will look at the creation of the abstract workflow and what could be improved. After this the ontology and instantiation of the operations will be discussed, followed by a closer look at the actual application. Finally, a discussion of the output of the application, the executable workflow, will be presented.

### 5.1 Limitations

During the creation and testing of the application several choices were made, which provides some relevant context for the results. Due to time constraints only one workflow was tested, this means that the results presented here are valid for this specific workflow, and are not necessarily valid in a broader context. Having said that, the ontology is created in such a way that the addition of more geo-operations does not change the way the application functions, nor would changing the number or order of the operations change the way the application creates the executable workflow. Furthermore, the ontology consists of five geo-operations, which can be seen as a limited amount considering the full operation library on offer in the execution engines. However, this does not necessarily take anything away from the validity of the results, as both raster and vector operations have been included and therefore both datatypes have been included in the testing. Adding several more operations would not add to the validity of the results, where adding a significant amount is beyond the possibilities of this research. Finally, the testing has occurred using two different execution engines, ArcMap and ILWIS. Even though a broader range of execution engines would provide a better insight into the possibilities of the Semantic Web as a tool to standardise operations and convert abstract to executable operations, the software used does provide some insight into the differences between fully developed software and software that is under development as well as open source versus proprietary software.

### 5.2 The abstract workflow

The abstract workflow that serves as an input for the application is now a JSON file, created by ILWIS model builder and slightly edited manually in order to lose some excess metadata which was no longer relevant. A first limitation of this input is the fact that it is not in the RDF format. It has been a consideration to translate the example workflow to RDF based on the OPMW and related ontologies as described in chapter 3, however the ontologies involved proved to be too complicated to easily recreate the abstract workflow in this format. Furthermore, as there are no tools that can be used, it would create a far steeper learning curve for people wanting to use the application created in this thesis. The abstract workflow as produced by ILWIS however, also has some downsides, since a workflow based on the workflow ontologies uses URIs to refer to operations rather than literals. This means that there is no possibility of confusion or misspelling of operation names, and the relevant parameters per operations are already defined. Next to this, this means that the queries that have to be created by the application are significantly simpler as the right URIs and parameters are already defined. So merely some properties of the already defined entity have to be loaded in order to get the descriptions of expected inputs. However, even though this would make the process easier, the current format of the workflow did not limit the proof of concept in the sense that it could still show

the functioning of the ontology and the conversion engine. Furthermore, ideally the application would accept different types of abstract workflows allowing for example for the use of simple visual workflow builders to allow easier use of non-experts in the field of geo-information.

For this proof of concept to function based on this abstract workflow it turned out that basic workflow logic cannot be omitted in the creation of a workflow. In the initial test of the application the workflow merely described the operations and their order, but didn't have the connection list as described in the previous chapter. While developing the application however, it turned out that without this logic a problem would arise as soon as multiple outputs of different operations would serve as the input for one of the next operations. This would then require the user to specify which outputs belong to which inputs. As the idea behind the application is to provide solutions to people with limited domain knowledge, the workflow should define this logic for them.

### 5.3 The ontology

As discussed in the previous chapter the Geo-Operation Ontology (Figure 5.1) is the core of the application, as it describes the requirements for execution of the operations in the workflow and provides the mapping to the executable python snippets per execution engine. Basically, it server as a way to standardise the operations in a manner that makes them execution engine independent. The process of creating the ontology went through several stages, as the intended standardisation of the operations through the ontology was hindered by the practical application. In the initial version of the ontology all operations were instances of their relative operation class (e.g. Clip Operation), defining the parameters for the operation in general. To then provide a mapping toward the specific python

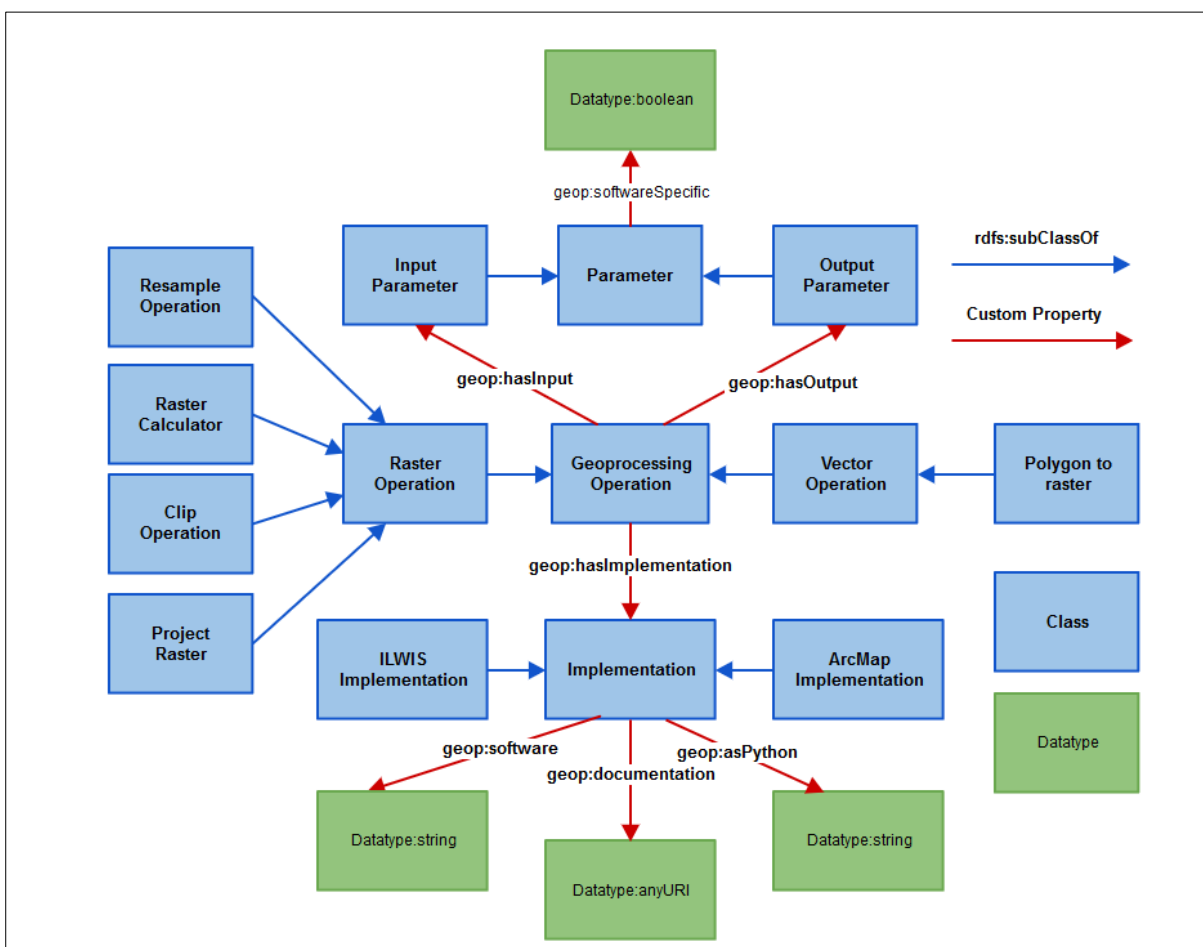


Figure 5.1: Geo-Operation ontology.

implementation two properties were created 'hasArcmapImplementation' and 'hasILWISImplementation'. However, as the initial workflows were created the inherent difference between the required inputs for both execution engines proved to be insurmountable. Therefore, the Implementation classes were created, allowing the instantiation of the operation on a lower level, describing the inputs specifically for the selected execution engine. This solution however has a downside. The description of the operations per specific execution engine significantly diminishes the measure of standardisation. In an effort to keep the description of operations at the highest possible level intact, and with the standardisation, the `geop:softwareSpecific` property was invented allowing the distinction between overlapping and execution engine specific parameters. Furthermore, as the connection between inputs and outputs of different operations as specified in the abstract workflow points to specific parameters, standardisation of these parameters was a minimum requirement for the application to function.

Further to the execution engine specific implementation of operations, the execution engines allow the setting of additional optional parameters. These parameters do not necessarily require a user input but do allow the user to finetune the output of the operation. For example, in the Polygon to Raster operation ArcMap allows the user to set a `cell_assignment` parameter which determines how the software deals with a cell that is occupied by two or more features. These optional parameters default to a standard value when nothing is specified. As the focus for this research lies mainly with the conversion of a software independent abstract workflow to an executable one the added value of adding the additional complexity of optional parameters to the operation instantiation was deemed outside the scope of this research. However, as it does play a part in the standardisation of operations in the ontology it is relevant to shortly discuss here. In the ontology an additional property of the Parameter class should be added; `geop:optional`. This would allow the user of the workflow or the application to opt for a standard or an expert version depending on the level of knowledge of the user.

#### 5.4 The Application

As the goal of this research is showcasing the added value of Linked Data in understanding and working with workflows the aim was to make the application as light as possible, putting most of the necessary logic in the ontology. This allows any other developer to easily create their own application around the ontology rather than having this logic hardcoded into an application. However, it was unavoidable to put part of the conversion logic into the application. First of all, the workflow logic, as extracted from the abstract workflow needed to be used by the application to create the executable workflow in the correct order. Such an element cannot be moved to the ontology as it is expected to be different for every workflow. In the application this workflow logic is used to circumvent human error, by using it to connect the input fields of the connected parameters, automatically matching the values in both fields, to make sure that the output name of one operation matches the input name of the connected operation.

When it comes to the actual conversion process quite some expert knowledge is required. In order to be able to output a correctly working executable workflow using the specific, in this case, python library, quite an extensive understanding of this library and the underlying software is needed. A large part of the time spent on developing the application went into setting up the correct output for the respective execution engines. As both execution engines, ArcMap and ILWIS differ quite a bit when it comes to user friendliness, level of documentation and level of development of their python libraries the largest effort went into understanding the ILWIS software. However, the ILWIS development team was available for support throughout the application development. A significant difference between the execution engines, when it comes to the level of development and user friendliness is that the ArcMap python interface, using the ArcPy library, provides functions such as `ListRaster`, which allow

```

import os
import fnmatch
from ilwis import *

#Function that creates a list from the rasters so that a process can easily be performed on all rasters
def CreateRasterList (workingDirectory, filename):
    rasterlist = []
    list = os.listdir(workingDirectory)
    for file in list:
        if fnmatch.fnmatch(file, '*' + filename + '*.asc'):
            rascov = RasterCoverage(file)
            rasterlist.append(rascov)
    return rasterlist

#Function that allows the creation of a georeference from an input raster
def createGeoreference (georef):
    rc = RasterCoverage(georef)
    georeference=rc.geoReference()
    return georeference

#set ilwis working directory
workingCatalog = 'file:/// ' + workingDirectory
Engine.setWorkingCatalog(workingCatalog)

```

*Listing 5.1: Code snippet to set up executable ILWIS workflow.*

```

import arcpy
    from arcpy.sa import *
    arcpy.CheckOutExtension("Spatial")
    arcpy.env.workspace = workingDirectory

```

*Listing 5.2: Code snippet to set up executable ArcMap workflow.*

for the adding of a list of rasters to an array in order to execute geo-operations for the entire array using a for loop. However, ILWIS does not provide such features, and needs manual writing of such functions. The creation of such functions and the importing of libraries needed for specific functions within each of the executable workflows requires a software specific setup of the executable workflow. The difference in required setup can be seen in Listing 5.1 and Listing 5.2.

As can be seen in Listing 5.1 there is a significant setup required for the executable ILWIS workflow. Currently this is hardcoded into the application as it sets up the file the rest of the code, the variables created based on user input and the python snippets from the ontology, is appended to. However, ideally this would be added to the ontology creating a dynamically created set up snippet based on the operations in the workflow. The code snippet above shows a function which extracts a georeference from an existing file calling an ILWIS function. This set up snippet is now hardcoded based on the required functionality for the operations. As more functions would be added to the ontology allowing a broader range of functions to be called in the application more of such python functions might be needed. Adding all the functions to every script would be superfluous so a logic would need to be added to the application in order to create this snippet dynamically. Furthermore, adding the logic this setup snippet needs to the ontology would allow the adding of new functionality or new software implementations by only editing the ontology, leaving the application as is. This would also allow to create new application based on the ontology as all required logic and information would be contained in the ontology.

Also in the setting up of the functions and the passing of data between the functions within the workflow there is a significant difference between ArcMap and ILWIS. Where ArcMap creates intermediate data by writing it to file, ILWIS stores intermediate data in a Python variable. This has some implications for the way data is prepared for the next step of the workflow. In the case of



ArcMap, as it stores all data in the working directory specified, the ListRaster function is reused at the start of every function, creating a new array of raster files as input for the operation. However, for ILWIS the procedure is different, as the intermediate data is stored in a variable, the data needs to be appended to the array as soon as the application is finished, in order for the function to not overwrite the result of the previous raster.

Even though there is a significant difference in the way the executable workflows are set up, the application is dynamic enough to be able to use the same function in creating the workflow for both execution engines. As it collects most information required from the ontology, except for the setup code which forms the start of the workflow, the SPARQL queries can be dynamically created obtaining the required code. This is done based on the inputs provided by the user, changing the operation and implementation for which information is requested based on the input and choices made by the user. The returned python snippets, are then written to a file based on the name provided by the user as output for the final operation in the workflow. This exemplifies the dynamism the ontology provides to the application.

### Data Preparation

For both raster and vector data there is a range of different data formats available, and most GIS have their own specific data formats on top of this range of common formats. This is also the case with both ArcMap and ILWIS. Where for the vector data used in this project the file format was ESRI Shapefile (.shp) which is useable in both GIS, the raster data used was in the ILWIS raster format MPR. Therefore, the data needed to be converted to be able to use the same data for both workflows. This was done using ILWIS which allows exporting the loaded raster files to the ArcInfo ASCII format (.asc), which is useable in both systems. After his conversion both systems were unable to use the data as in the conversion the projection information was lost. This was solved using a short python script calling the 'Define Projection' tool in ArcMap and defining the original WGS84 projection for all raster files.

Another bit of data preparation was necessary in order for the workflows to be able to be executed. In the reprojection of the rainfall data as part of the workflow, one of the parameters that needs to be defined is 'Output Projection' which is WGS84 UTM36S. Where ArcMap simply allows the input of two variables called 'cell size' and an EPSG code to indicate the required projection, ILWIS requires a more complicated input. It uses either a georeference file which contains this information or can extract this information from an existing raster file. In order for the output of the operation to be in the correct projection and cell size a raster file was created containing the required georeference. In order for the workflow to work in ILWIS the user input in the application should reference such a file, and therefore requires a prepared georeferenced file. This adds some complication to the process of executing the workflow in ILWIS, especially for the non-expert user.

### 5.5 The output

The output of the application is twofold, the initial result of running the application is the executable workflow. However, running the workflow should provide you with a solution to the initial spatial problem which required the workflow. Both warrant some discussion here.

The executable workflow in both cases is a stand-alone script. This means they can be executed straight from the desktop, using the related software's python library, and don't need to be executed from the python window in the relevant software. However, in the case of ArcMap there is the requirement to own the required license for the Spatial Analysis extension, which needs to be used. In the case of ILWIS this is not required as it is open-source software, and no licenses are required. The output in both cases is a Python script. The advantage here is that Python is an open-source language, which focusses on readability of the code. This means for non-expert users an easier

understanding of the output script that is produced, and it is arguably one of the easier languages if the end-user wants to make changes to the executable workflow. However, there is also a downside to using python, being that as soon as the query results from the triple store are written to the Python variables they lose their semantic enrichment. As the operations and relevant parameters are queried from the triple store the results are returned in a JSON format, as a string. Ideally the returned data would retain its semantic enrichment by creating a workflow in one of the RDF serialisations. The most logical format for this would be JSON-LD as discussed in chapter 2. As the ILWIS model builder already supports exporting JSON workflows, as demonstrated with the abstract workflow the step towards also allowing the import of JSON workflows should not be far away. As JSON-LD allows the parsing of the file as JSON, the workflows created by the application could be in the JSON-LD format, using the URIs from the ontology, retaining its semantic enrichment while at the same time being able to be read as JSON file and executed in this manner through, for example, ILWIS.

The second part of the output is the actual results obtained by executing the executable workflows in their respective execution engines. Important to note here is that a difference in the outputs is to be expected. The difference in parameters, especially in the case of ArcMap, default choices for optional parameters already means that slight, and possibly significant, differences in the output are to be expected. This automatically means that in case of reusing scientific workflows a conscious choice needs to be made in case of all the optional parameters, and the current way of working would not suffice. This would require an extension of both the ontology and the abstract workflow. However, in the case of the non-expert user, requiring a solution to his spatial problem these problems might be of a lower magnitude. For the execution of the workflows raster datasets describing rainfall were used, ranging over a period of time from the end of October 2014 – start of June 2015, which is the period when rain can be expected in the MaMaSe project area. The execution of the workflows produced the following results.

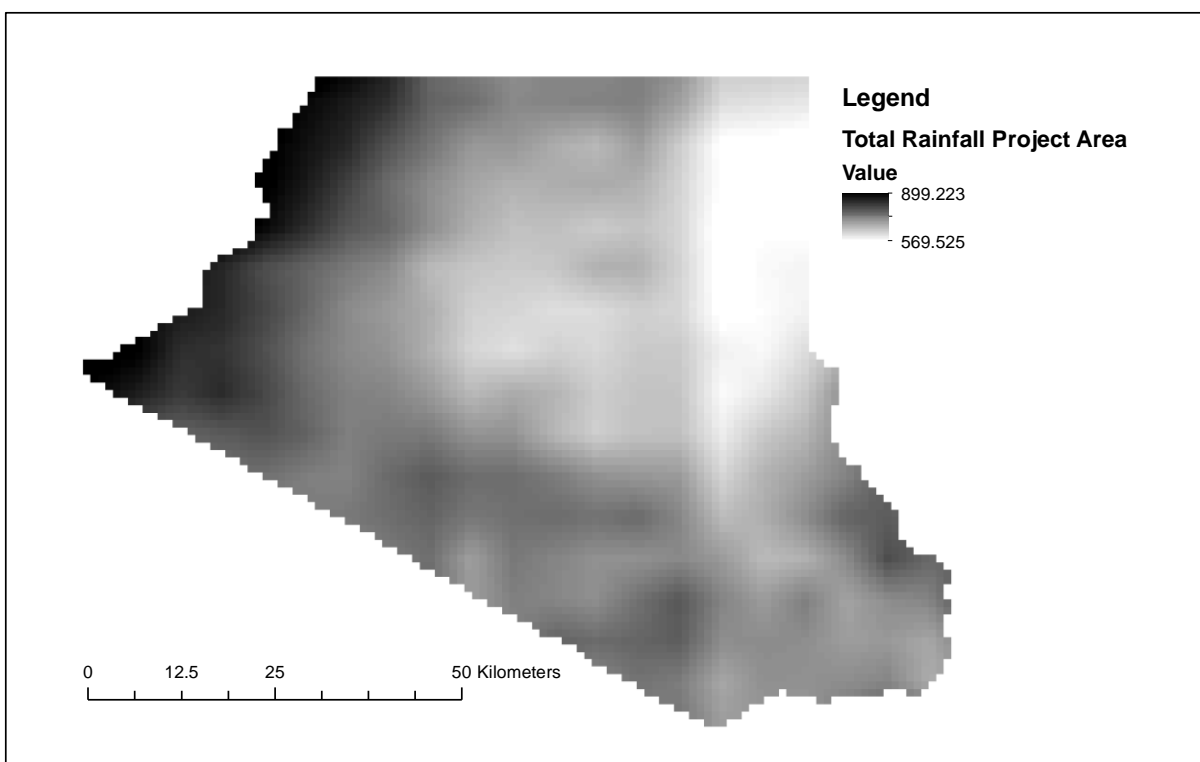


Figure 5.2: Output ILWIS Workflow.

When comparing Figure 5.2, the output when executing the ILWIS workflow, and Figure 5.3, the output when executing the ArcMap workflow, a slight difference in result is noticeable. The lower limit of the total rainfall in the rain season in the MaMaSe project area has a difference of 2.9mm where the upper limit has a difference of 1.3mm. The fact that these differences exist means that the expected difference as stated before exists. However, the significance of this difference, when looking at the potential non-expert user, is marginal. When, for example, an African farmer who is trying to find the wettest area in the region to grow certain crops, a difference of a few mm is irrelevant when the range of rainfall difference over the total area is approximately 330mm.

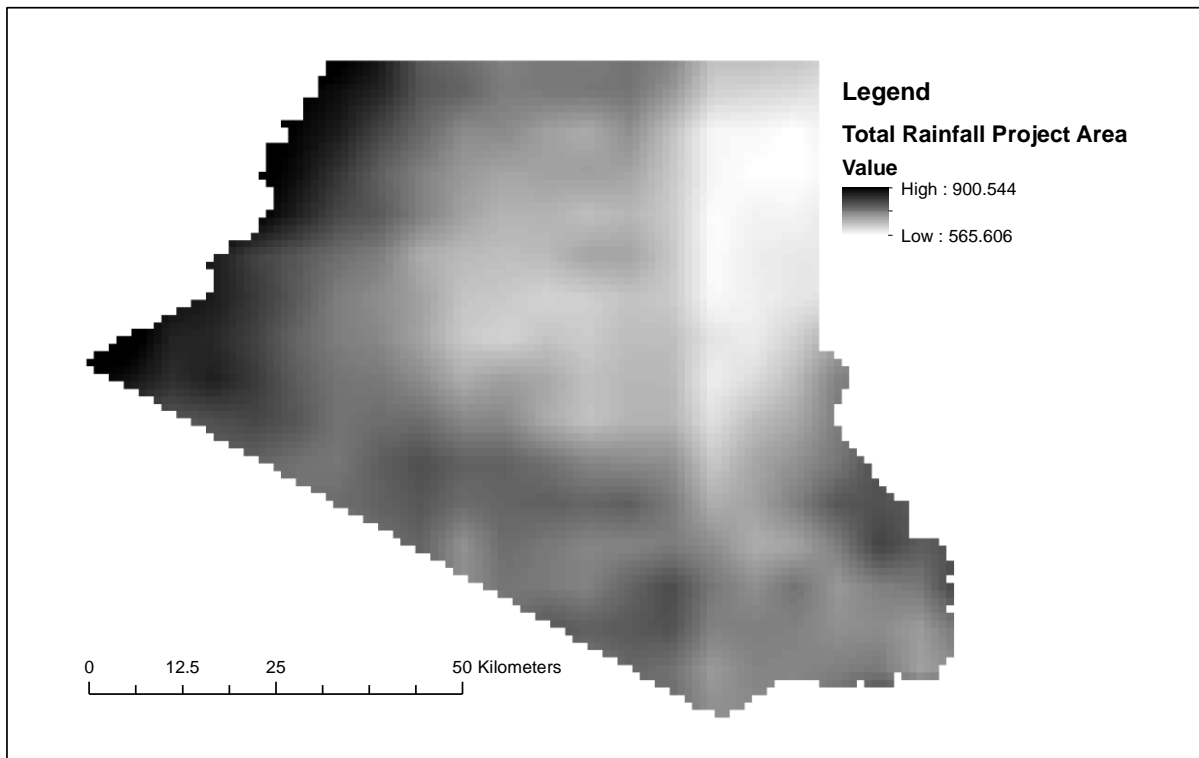


Figure 5.3: Output ArcMap Workflow.

A note that has to be made when discussing these results is the output of the ArcMap workflow has been slightly altered for better comparison. A difference in the way the clip operation is executed by both execution engines makes that it produces a different result when it comes to the area taken into consideration. Where the ILWIS operation takes an approach, which uses Boolean logic to perform the clip, meaning that anything within the border is true and is added to the output map, anything outside is false and is discarded, ArcMap uses a bounding box. This bounding box is drawn around the project area and the entire rectangle is then used to clip the rainfall raster. This means that also the output of the workflow is the shape and size of this bounding box rather than the shape and size of the project area. In order to properly compare the two results, another clip operation, using the original boundary Shapefile has been performed, resulting in the output in Figure 5.3. The abstract workflow specifically asks for a conversion of this vector boundary to raster, resulting in the 'incorrect' result for the ArcMap workflow. Using the vector-based clip was therefore not an option in the actual workflow as this would change the abstract workflow and make it a software specific workflow, which defeats the purpose of the application. However, additional logic added to the application,

understanding the way the specific execution engines execute certain operations, should be able to resolve this difference without changing the abstract workflow.

## 6 Conclusions and recommendations

This last chapter will start with a short summary of the most important conclusions followed by a step by step discussion of all research objectives and research questions as stated in section 1.3. The main research objective is:

*The development of a method to semantically enrich an abstract workflow and convert it to an executable workflow using Semantic Web technologies.*

The emphasis in this research was on discovering the added value of the Semantic Web in sharing and understanding (abstract) workflows. The comprehension of these workflows should lead to easier use and reuse by non-experts. The aim was to achieve this goal by semantically enriching abstract workflows and through standardisation of the geo-operations achieve easy conversion to executable workflow formats.

### 6.1 Conclusions

The research aim implied a search for a way to programmatically interpret the different steps and logic of an abstract workflow. Using the Semantic Web as a tool to do so, making it understandable to both machines and humans, allowing them to work in unison in providing the additional parameters to convert the workflow to executable. The role the Semantic Web plays in this is very much one of standardisation of the semantics of the different geo-operations and their parameters, next to providing all this in an online, interoperable and free to use format allowing it to be used by anyone, without requiring any expert knowledge.

This thesis proved that this is certainly possible. The abstract workflow was created in an open source GIS, which produced the workflow in JSON, a language which can be parsed in many programming languages. Using a Semantic Web ontology semantics were provided for the different geo-operations, in a software independent way, providing the necessary documentation for the users to understand the required inputs. These human inputs are used by an application to create a software specific executable workflow based on the selected execution engine by the user. This application has been developed using Node.js which is server-side JavaScript which allows this application to run online. As all different elements of this thesis can be hosted and run online, with the possibility to execute the resulting executable workflow in an open source GIS, it allows use by anyone with an internet connection. This proves that the Semantic Web is of added value in solving spatial problems, especially in areas where expert knowledge is often missing, or an expensive commodity. Allowing the workflows created by one, to be easily reused and understood world-wide, especially in developing countries provides a step forward in reusability of workflows, but also in allowing people world-wide to more easily come to a solution to their spatial problems.

While concluding that this step forward is certainly possible and the role of the Semantic Web in this could potentially be a significant one, there is also still a considerable effort required to achieve this. A first step required is the extension of the ontology to include a broad range of geo-operations. Even though this research showed that the ontology is a useful way to take a step towards standardisation of geo-operations, compromises had to be made during the development of the ontology in order to accommodate the differences in implementation between the execution engines. This requires additional research to find a way to bridge these differences. Another conclusion that can be drawn from based on the disproportionate amount of effort spend on creating the proof of concept is that the creation of the executable code snippets for each operation, needed for the conversion to executable, requires quite some expert knowledge. Where the ontology could potentially be partially

crowdsourced among domain experts, the mapping to executable code is a significantly complicated process that the developers of the relevant execution engine should be involved.

### Research objectives and questions

This section will provide a discussion of whether the research objectives have been achieved and answer the questions posed per research objective.

#### *Objective 1: Understanding the way the Semantic Web and Linked Data work.*

This objective focussed on literature research aimed at understanding how the Semantic Web works and what it can contribute to understanding and formalizing domain specific knowledge. This has been achieved and described in chapter 2.

##### *1.1 How are data sources connected on the Semantic Web?*

- Linked Data is the technology behind the Semantic Web used to connect data sources based on the Linked Data principles; using HTTP URIs as names for things, using Semantic Web standards and linking to other URIs to link the data.
- Data is described using the Resource Description Framework (RDF) storing data in triples creating knowledge graphs. By connecting these knowledge graphs, the aim is to turn the web into a web of data.
- Ontologies are used to create rules for the triple structure and the logic of the data models. Ontologies allow for standardisation by serving as an explicit data model, which others can adhere to, allowing their data to be described using the same concepts, facilitating linkage between different datasets.

##### *1.2 What tools are necessary to be able to use this technology?*

- To create Linked Data ontologies, development tools should be used to significantly reduce the effort required. Such a development tool is Protégé. This tool creates ontologies based on the OWL and RDF(S) standards. Furthermore, it can be used to create individuals in the created classes, so allowing to create the data model and data in the same tool.
- To store the Linked Data triple stores are used. There is a wide offering of different triple stores. From a GI perspective a relevant feature that distinguishes between them is the implementation of GeoSPARQL to be able to create query filters based on topological relations.
- Queries to the triple store are done using the SPARQL standard. These can be performed from any programming language with access to the internet. All it takes is an HTTP get request, with the query as a parameter.

##### *1.3 What ontologies are available?*

- Basic ontologies to create Linked Data and ontologies; RDF, RDFS & OWL. These provide the basis to create classes and properties. OWL specifically further allows the use of restrictions to add some logic to the data models.
- Workflow ontologies: PROV and OPM are ontologies that can be used to describe data provenance. This allows the description of operations that have been performed and changes made to the data. This is only suitable for description of completed processes, so executed workflows. These are extended by the P-Plan ontology which allows the description of a *Plan*

allowing the description in RDF of a executable workflow. OPMW bring these together in one ontology and adds the description of *Templates* which are abstract workflows.

- Ontologies describing geo-operations are currently lacking, and had to be created for this thesis.

#### *1.4 How can ontologies/scripts/workflows designed during the research be stored/accessed on the Semantic Web?*

- Data created on the Semantic Web, be it workflows, ontologies or knowledge graphs are stored in triple stores.
- This data is accessible through SPARQL, the query language of the Semantic Web. In order to extract this data from the triple store the application created in this thesis creates dynamic queries, using a HTTP GET request to obtain the data.

#### *Objective 2: Assess different workflow design and scripting languages on their potential connectivity to the Semantic Web.*

The research done for objective 1 revealed that the Semantic Web is in the largest sense similar to the current web and allows for any scripting language or workflow design language that can be interpreted by the regular web languages to be used in collaboration with the Semantic Web. This objective has been accomplished and described in chapter 3.

#### *2.1 Which scripting languages, workflow design tools or standards are available for Geo Information workflows?*

- Standards for workflows are available, but interoperability between different workflow management systems is limited. Therefore, a standardisation effort is required, which has been partially accomplished by the workflow ontologies described in objective 1.
- Mostly GI workflows are created by the same software that executes these workflows, so no specific workflow builders separate from the GIS themselves. However, python libraries such as Arcpy do allow the creation of workflows outside of the GIS client. But still require the specific ArcMap license to execute. So, still no interoperability.
- The main scripting language for GIS workflows is Python, however, both GIS taken into account in this research both require their own Python library to execute operations.

#### *2.2 Do they allow for direct connectivity to the Semantic Web or is conversion or additional software needed?*

- The Semantic Web uses URIs to refer to things, as does the ontology describing the geo-operations. As this is not incorporated into the model builders, there is a conversion required in the form of dynamically created SPARQL-queries to 'connect' the workflow to the Semantic Web.
- Additional software is required in the form of an application that interprets the workflow and enriches it with semantics, as developed in this research.

#### *2.3 What software is suitable to construct the necessary workflows?*

- In order to allow the semantic enrichment of the abstract workflow, the workflow needs to be exported in a format that allows a web-connected language to parse the workflow and

extract the logic and operations. In this case suitable software is any model builder that allows export to easily interpretable languages such as JSON.

- ILWIS model builder has been used as it creates a JSON output which can be interpreted in the JavaScript application used in this research. This application in its turn is used to then create the executable workflow based on user input.

*Objective 3: Assess the state of the art of the Semantic Web, to review the applicability of the selected use case to the proof of concept.*

As little to no ontologies were available describing geo-operations in a relevant way to use in the semantic enrichment of abstract workflows, this objective was achieved in the sense that the state of the art, when talking about available ontologies was not sufficient. However, the state of the art when it comes to easily useable and accessible ontology development tools proved sufficient, as an own geo-operation ontology was created. This automatically means that any workflow could be applied to the proof of concept as the RDF descriptions of the relevant geo-operations had to be created for the specific workflow.

*3.1 Are relevant ontologies available to describe the workflow and geo-processes*

- Relevant ontologies describing geo-operations, making certain workflows more suitable for use in the proof of concept than others, were not available.

*3.2 If ontologies are not available, is it possible to create own ontologies?*

- Creating a basic proof of concept ontology is possible using existing and well documented ontology development tools, in this case Protégé.
- Ontology has been developed using existing and well documented ontology languages. RDF, RDF Schema and the Web Ontology Language (OWL).
- Creating an own ontology proved possible for a relatively inexperienced user of the Semantic Web technologies.

*3.3 Is the complexity of the selected workflow sufficient to assess the added value of the Semantic Web?*

- The selected workflow consists of five different operations, of which two (resample & reproject) are available as one operation in both GIS used. Therefore, they have both been described in the geo-operation ontology, but only one has been used in the workflow.
- This workflow was selected based on both reasonable size, making test execution possible in a suitable time frame and incorporation of both vector and raster data. The workflow also includes different types of operations (conversion, sub-setting, aggregation) making it sufficiently complex to experiment with different aspects relevant in GI workflows.
- Extending beyond this amount of operations would extend execution time when testing the workflows without directly adding value to the outcomes of the research.

*3.4 Who are the expected users of the proof of concept?*

- The expected users of the application are non-expert users, wanting to solve spatial problems they have by reusing workflows created by experts. However, as the proof of concept is not



yet able to provide parameter suggestions, but only guidance to the users, some level of experience is still required. Additional development should allow use by users that are completely inexperienced in the field of GIS.

- Therefore, as much as possible has been automated, minimizing the user input required to reach the point of execution.

#### *Objective 4: Assess the creation process of the proof of concept and its success.*

The creation of the proof of concept has been a process that has taken considerably more effort than was anticipated. Significant differences in implementation of geo-operations led to more effort in both the standardising effort through the ontology as well as mapping the operations to executable code.

##### *4.1 How did the selected tools and standards influence the proof of concept?*

- As the goal of the application was to allow use by non-experts world-wide, suitable tools and languages had to be open source and where possible platform independent.
- Selected languages are Python, JavaScript and Node.js, all platform independent and free to use.
- For the development of the application NetBeans IDE has been selected, available on multiple platforms and also free to use.
- The tools and languages used in no way diminished the success of the proof of concept. However, the fact that the creation of the abstract workflow was based on operation names rather than URIs added some complexity to the interpretation of the workflow.

##### *4.2 Did the proof of concept provide relevant output to assess the success of the proof of concept?*

- The created output is twofold. 1) An executable workflow 2) The output of the workflow in the form of a raster map. Both outputs were as expected and useful in determining the success of the proof of concept.
- Both executable workflows can directly be executed as produced by the application and execute without failure.
- Both outputs of the workflow are similar, with insignificant differences of a few millimetres of rainfall over the entire rain season, at least for non-expert users. However, the ArcMap output needed an additional clip operation to make a useful comparison to the ILWIS output, because of a different implementation of the clip operation used in the workflow.

##### *4.3 How can the proof of concept be tested with probable users?*

- Actual testing with probable users was not achieved in this research. The technical implementation of the proof of concept took up considerable time, not leaving time or resources to set up proper user testing.
- User testing could be achieved by providing non-expert users with abstract workflows and seeing if the application guides them sufficiently in their user input to achieve successful execution of the executable workflow, leading to the expected output raster map.

**Objective 5: Analyse the results of the proof of concept and provide recommendations towards further research.**

The proof of concept application provides the expected results, so this objective has been accomplished. As already discussed in the previous objective the proof of concept application output workflow executes in a satisfactory manner and both produce similar results. Therefore, the aim of a software independent way of describing workflows has been achieved. Further standardisation in the ontology is desirable as discussed in section 5.3.

*5.1 Which elements of the system work as intended and which should be improved?*

- The semantic enrichment of the abstract workflow and the conversion to executable work as intended. This is not to say that there aren't any compromises made in the creation of the proof of concept. Several of these are discussed in chapter 5.
- The ontology provides less standardisation than was intended, as part of the operations had to be described on an implementation level, rather than as a standard operation fully software independent and interoperable.

*5.2 Are any limitations caused by the selected tools or are there issues with the underlying technology?*

- The ontological limitations mentioned are inherent to the differences found in implementation of operations in GIS. Therefore, the selection of tools, or the technology used, could not have circumvented this, so the selected tools and technology were adequate.

*5.3 What steps are necessary to apply this system on a larger scale or within organisations?*

- Extension of the ontology to include more operations, the current ontology is rather limited in its possibilities.
- Extension of the ontology to provide mappings to executable code for more different execution engines, however with an open source solution in ILWIS and a popular proprietary GIS client in ArcMap, this is not required for large scale use.
- Further development of the explicit semantics of the variables to further guide the user in selecting the appropriate inputs. For example, defining mandatory data types for the input parameters.

## 6.2 Recommendations

This thesis has provided a proof of concept, showing the role the Semantic Web can play in solving spatial problems. However, there is still much research to be done to see if the Semantic Web can indeed play an integral part in solving spatial problems. This section provides some recommendations for future research directions which in the eyes of this researcher should be explored next.

### **User testing**

Even though this thesis proved that the Semantic Web can play a part in helping non-expert users solve their spatial problems, this has not been definitively proven as no user testing has occurred with actual non-expert users. In order to improve the application created, testing with potential users should occur, based on which the application can be further improved.

### **Extension of ontology for full operations support**

The current ontology is a mere prove of concept, providing a first suggestion towards a full geo-operation ontology. Extending this ontology is a separate research direction all together. Requiring extensive research into the implementation differences of geo-operations in different execution engines. But also, their overlap, as this is where the standardisation effort should start, as was done in this research by specifying the difference between general and software specific parameters.

### **Creating abstract workflows**

In this research an abstract workflow was created from an existing one, making it more abstract to serve as the required input for the proof of concept. However, if the application should be used by non-expert users, a relevant research direction would be the creation of abstract workflows by non-expert users as well. This preferably would not occur in a GIS model builder, but rather in a more abstract environment, allowing the user to use its own terms rather than software specific operations names. Alternative names for operations could be added to ontologies using the SKOS standard, which allows for multiple alternative labels for instances.

### **Advanced operation logic**

The operation logic in the abstract workflow used in this research is simply based on the connection of output and input parameters of the different operations. During the interpretation of the workflow by the application there is limited understanding of the relationship between operations. However, as concluded in the previous section the step of converting the boundary file to raster before performing the clip operation resulted in an unwanted result in ArcMap, where using the vector layer as input directly would have provided a better result. Advanced description of operation inputs and requirements and understanding the specific implementation of an operation in an execution engine, could allow the application to suggest removing the 'convert to raster' application in case of execution in ArcMap.

### **Online execution of workflows**

Where the entire application, ontology and data can be hosted online, the execution in this research still occurs in an offline environment. However, with the development of WPS, this execution could potentially also occur online. Removing any hardware requirements for the potential user. As discussed in chapter 3, previous research in this direction has been done by Diniz (2016), and can be used as a starting point for research in this direction.

## Bibliography

- Aasman, J., & Cheetham, K. (2011). RDF browser for data discovery and visual query building. In *Proceedings of the Workshop on Visual Interfaces to the Social and Semantic Web (VISSW 2011)*. (pp. 1–4).
- Allemang, D., & Hendler, J. (2008). *Semantic web for the working ontologist : modeling in RDF, RDFS and OWL*. *Journal of empirical research on human research ethics JERHRE* (Vol. 6). <http://doi.org/10.1525/jer.2011.6.3.toc>
- Barga, R., & Gannon, D. (2007). Scientific versus business workflows. In *Workflows for e-Science: Scientific Workflows for Grids* (pp. 9–16). London: Springer. [http://doi.org/10.1007/978-1-84628-757-2\\_1](http://doi.org/10.1007/978-1-84628-757-2_1)
- Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., ... Goble, C. (2013). Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2), 599–611. <http://doi.org/10.1016/j.future.2011.08.004>
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34–43.
- Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3), 1–22. Retrieved from <http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf>
- Brauner, J. (2015). *Formalizations for geoperators-geoprocessing in Spatial Data Infrastructures*. Dresden University of Technology. Retrieved from [http://www.qucosa.de/recherche/frontdoor/?tx\\_slubopus4frontend\[id\]=18250](http://www.qucosa.de/recherche/frontdoor/?tx_slubopus4frontend[id]=18250)
- Ciocoiu, M., Gruninger, M., & Nau, D. S. (2000). Ontologies for Integrating Engineering Applications. *Journal of Computing and Information Science in Engineering*, 1(1), 12–22. <http://doi.org/10.1115/1.1344878>
- DBpedia.org. (2017). About. Retrieved February 4, 2017, from <http://wiki.dbpedia.org/about>
- Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., ... Wenger, K. (2015). Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46, 17–35. <http://doi.org/10.1016/j.future.2014.10.008>
- Diniz, F. D. E. C. (2016). *Composition of Semantically Enabled Geospatial Web Services*. University of Twente.
- European Commission (2007). Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE).
- European Parliament and Council (2003). Directive 2003/98/EC of 17 November 2003 on the re-use of public sector information. OJ L, 345, 90.
- Fang, F. C., & Casadevall, A. (2011). Retracted science and the retraction index. *Infection and Immunity*, 79(10), 3855–3859. <http://doi.org/10.1128/IAI.05661-11>
- Ferre, S. (2017). Sparklis: An Expressive Query Builder for SPARQL Endpoints with Guidance in Natural Language. *Semantic Web*, 8(3), 405–418. <http://doi.org/10.3233/SW-150208>
- Garijo, D. (2015). *Mining Abstractions in Scientific Workflows*.
- Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., & Goble, C. (2014). Common motifs in

- scientific workflows: An empirical analysis. *Future Generation Computer Systems*, 36, 338–351. <http://doi.org/10.1016/j.future.2013.09.018>
- Garijo, D., & Gil, Y. (2012). *Towards Open Publication of Reusable Scientific Workflows: Abstractions, Standards and Linked Data. Internal Project Report*. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.222.9322>
- Garijo, D., Gil, Y., & Corcho, O. (2017). Abstract, Link, Publish, Exploit: An End to End Framework for Workflow Sharing. *Future Generation Computer Systems*. <http://doi.org/10.1016/j.future.2017.01.008>
- GeneOntologyConsortium. (2004). The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research*, 32(Database issue), 258–261. <http://doi.org/10.1093/nar/gkh036>
- Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., ... Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, 40(12), 24–32. <http://doi.org/10.1109/MC.2007.421>
- Gil, Y., Ratnakar, V., & Deelman, E. (2007). Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the National Conference on Artificial Intelligence (Vol. 22, No. 2)* (pp. 1767–1774). Retrieved from <https://www.aaai.org/Papers/AAAI/2007/AAAI07-284.pdf>
- Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., ... de Roure, D. (2010). myExperiment: A repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, 38(SUPPL. 2), 677–682. <http://doi.org/10.1093/nar/gkq429>
- Goecks, J., Nekrutenko, A., & Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8), R86. <http://doi.org/10.1186/gb-2010-11-8-r86>
- Heath, T., & Bizer, C. (2011). *Linked data: Evolving the Web into a global data space (1st edition). Synthesis Lectures on the Semantic Web Theory and Technology 11* (Vol. 1). <http://doi.org/10.2200/S00334ED1V01Y201102WBE001>
- Hu, Y., & Li, W. (2017). *Spatial Data Infrastructure. The Geographic Information Science & Technology Body of Knowledge*. Retrieved from <http://dx.doi.org/10.22224/gistbok/2017.2.1>
- Iwaniak, A., Kaczmarek, I., Strzelecki, M., Lukowicz, J., & Jankowski, P. (2016). Enriching and improving the quality of linked data with GIS. *Open Geosciences*, 8(1), 323–336. <http://doi.org/10.1515/geo-2016-0020>
- Janssen, K. (2011). The influence of the PSI directive on open government data: An overview of recent developments. *Government Information Quarterly*, 28(4), 446–456. <http://doi.org/10.1016/j.giq.2011.01.004>
- Lin, C., Lu, S., Lai, Z., Chebotko, A., Fei, X., Hua, J., & Fotouhi, F. (2008). Servic-Oriented Architecture for VIEW: a Visual Scientific Workflow Management System. In *IEEE International Conference* (pp. 335–342). <http://doi.org/10.1109/SERVICES.2007.69>
- Liu, J., Pacitti, E., Valduriez, P., & Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, 13(4), 457–493. <http://doi.org/10.1007/s10723-015-9329-8>
- Lohmann, S., Link, V., Marbach, E., & Negru, S. (2014). WebVOWL: Web-based visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management* (pp. 154–158). Springer International Publishing. <http://doi.org/10.1007/978-3->

- Ma, X. (2015). Geoinformatics in the Semantic Web. In *Proceedings of the 17th Annual Conference of the International Association for Mathematical Geosciences (IAMG 2015)* (pp. 18–26).
- Maali, F., Erickson, J., & Archer, P. (2014). Data catalog vocabulary (DCAT). *W3C Recommendation*, 16.
- Mamase.org. (2016). Mamase | About. Retrieved February 18, 2016, from <http://www.mamase.org/mamase/about>
- Mates, P., Santos, E., Freire, J., & Silva, C. T. (2011). CrowdLabs: Social analysis and visualization for the sciences. In *International Conference on Scientific and Statistical Database Management* (pp. 555–564). [http://doi.org/10.1007/978-3-642-22351-8\\_38](http://doi.org/10.1007/978-3-642-22351-8_38)
- Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., & Paulson, P. (2011). The Open Provenance Model. *Future Generation Computer Systems*, 27(6), 743–756. [http://doi.org/10.1007/978-3-540-89965-5\\_31](http://doi.org/10.1007/978-3-540-89965-5_31)
- OGC (2012). OGC GeoSPARQL - A Geographic Query Language for RDF Data. OGC Implementation Standard. Open Geospatial Consortium, September 2012.
- Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., ... Li, P. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17), 3045–3054. <http://doi.org/10.1093/bioinformatics/bth361>
- Oliveira, W., de Oliveira, D., & Braganholo, V. (2015). Experiencing PROV-Wf for provenance interoperability in SWfMSs. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8628, 294–296. [http://doi.org/10.1007/978-3-319-16462-5\\_38](http://doi.org/10.1007/978-3-319-16462-5_38)
- Patroumpas, K., Giannopoulos, G., & Athanasiou, S. (2014). Towards GeoSpatial Semantic Data Management : Strengths , Weaknesses , and Challenges Ahead. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. (pp. 301–310). <http://doi.org/10.1145/2666310.2666410>
- Rodriguez, M. A., Bollen, J., Gershenson, C., & Watkins, J. H. (2012). Using RDF to Model the Structure and Process of Systems. In *Unifying Themes in Complex System VII* (pp. 222–230). Retrieved from <http://neeci.edu/events/iccs7/papers/1012f24806523020a58f28ddbde2.pdf>
- Salado-Cid, R., & Romero, J. R. (2016). Enabling the Definition and Reuse of Multi-Domain Workflow-Based Data Analysis. In *International Conference on Intelligent Systems Design and Applications* (pp. 684–696). <http://doi.org/10.1007/978-3-319-53480-0>
- Schätzle, A., Przyjaciół-Zablocki, M., Skilevic, S., & Lausen, G. (2015). S2RDF: RDF Querying with SPARQL on Spark. In *Proceedings of the VLDB Endowment* (pp. 804–815). <http://doi.org/10.14778/2977797.2977806>
- Scheider, S., Degbelo, A., Lemmens, R., Van Elzakker, C., Zimmerhof, P., Kostic, N., ... Banhatti, G. (2017). Exploratory querying of SPARQL endpoints in space and time. *Semantic Web*, 8(1), 65–86. <http://doi.org/10.3233/SW-150211>
- Simmhan, Y. L., Plale, B., & Gannon, D. (2005). A Survey of Data Provenance in e-Science. *ACM SIGMOD Record*, 34(3), 31–36. <http://doi.org/10.1145/1084805.1084812>
- Sonntag, M., Karastoyanova, D., & Deelman, E. (2010). Bridging the Gap Between Business and Scientific Workflows: humans in the loop of scientific workflows. In *IEEE Sixth International Conference on e-Science* (pp. 206–213). Retrieved from [bitpipe.com](http://bitpipe.com)

- Steiniger, S., & Hunter, A. J. S. (2013). The 2012 free and open source GIS software map - A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, 39, 136–150. <http://doi.org/10.1016/j.compenvurbsys.2012.10.003>
- Vandenbroucke, D., Zambon, M. L., Crompvoets, J., & Dufourmont, H. (2008). INSPIRE Directive: Specific requirements to monitor its implementation. *A Multi-View Framework to Assess SDIs*, 327.
- Veeckman, C., Jedlička, K., De Paepe, D., Kozhukh, D., Kafka, Š., Colpaert, P., & Čerba, O. (2017). Geodata interoperability and harmonization in transport: a case study of open transport net. *Open Geospatial Data, Software and Standards*, 2(1), 3. <http://doi.org/10.1186/s40965-017-0015-6>
- Vilches-blázquez, L. M., Villazón-terrazas, B., Saquicela, V., & León, A. De. (2010). GeoLinked Data and INSPIRE through an Application Case, 446–449.
- W3C. JSON-LD Primer. A Context-based JSON Serialization for Linked Data. Draft Community Group Report. World Wide Web Consortium, October 2017.
- W3C. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, World Wide Web Consortium, December 2012.
- W3C. PROV Model Primer. W3C Working Group Note. World Wide Web Consortium, April 2013.
- W3C. RDF 1.1 Primer. W3C Working Group Note, World Wide Web Consortium, June 2014.
- W3C. RDF Schema 1.1. W3C Recommendation. World Wide Web Consortium, February 2014.
- W3C. SPARQL 1.1 Query Language. W3C Recommendation, World Wide Web Consortium, March 2013.

## Appendices

### Appendix 1: Geo-operation ontology

@prefix : <http://www.semanticweb.org/ontology/GeoOperations#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

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

@prefix xml: <http://www.w3.org/XML/1998/namespace> .

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

@prefix geop: <http://www.semanticweb.org/ontology/GeoOperations#> .

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

@base <http://www.semanticweb.org/ontology/GeoOperations> .

```
<http://www.semanticweb.org/ontology/GeoOperations> rdfs:type owl:Ontology ;
              rdfs:label "Geo-operation ontology" ;
              rdfs:comment "This ontology provides a structure for describing geo-
operations, and their implementations in different execution engines" .
```

# Annotation properties

```
### http://www.w3.org/2000/01/rdf-schema#comment
```

```
rdfs:comment rdfs:range xsd:string ;
              rdfs:domain geop:GeoOperation ,
              geop:Implementation ,
              geop:Parameter .
```

```
### http://www.w3.org/2000/01/rdf-schema#label
```

```
rdfs:label rdfs:range xsd:string ;
           rdfs:domain geop:GeoOperation ,
           geop:Implementation ,
           geop:Parameter .
```

# Object Properties

```
### http://www.semanticweb.org/ontology/GeoOperations#hasImplementation
```

```
geop:hasImplementation rdfs:type owl:ObjectProperty ;
                        rdfs:domain geop:GeoOperation ;
                        rdfs:range geop:Implementation ;
                        rdfs:comment "Points towards an entity of the implemenation class describing the
details for the execution of this operation in a specific execution engine" ;
                        rdfs:label "Has Implementation" .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#hasInput
```

```
geop:hasInput rdfs:type owl:ObjectProperty ;
              rdfs:domain geop:GeoOperation ;
              rdfs:range geop:Input ;
              rdfs:comment "Specifies the input for a geo-operation" ;
              rdfs:label "Has an input" .
```



```
### http://www.semanticweb.org/ontology/GeoOperations#hasOutput
geop:hasOutput rdf:type owl:ObjectProperty ;
    rdfs:domain geop:GeoOperation ;
    rdfs:range geop:Output ;
    rdfs:comment "Specifies the output for a geo-operation" ;
    rdfs:label "Has an output" .
```

## # Data properties

```
### http://www.semanticweb.org/ontology/GeoOperations#asPython
geop:asPython rdf:type owl:DatatypeProperty ;
    rdfs:domain geop:Implementation ;
    rdfs:range xsd:string ;
    rdfs:comment "A piece of python code needed to execute this operation in the specified
software"^^xsd:string ;
    rdfs:label "Python Script"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#documentation
geop:documentation rdf:type owl:DatatypeProperty ;
    rdfs:domain geop:Implementation ;
    rdfs:range xsd:anyURI ;
    rdfs:comment "The documentation provided by the developer of the execution
engine"^^xsd:string ;
    rdfs:label "Documentation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#optional
geop:optional rdf:type owl:DatatypeProperty ;
    rdfs:domain geop:Parameter ;
    rdfs:range xsd:boolean ;
    rdfs:comment "Indicates whether a parameter is specific to an execution engine or valid for
all execution engines" ;
    rdfs:label "Software Specific"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#software
geop:software rdf:type owl:DatatypeProperty ;
    rdfs:domain geop:Implementation ;
    rdfs:range xsd:string ;
    rdfs:comment "Execution engine in which this operation can be executed"^^xsd:string ;
    rdfs:label "Software"^^xsd:string .
```

## # Classes

```
### http://www.semanticweb.org/ontology/GeoOperations#ArcMapImplementation
geop:ArcMapImplementation rdf:type owl:Class ;
    rdfs:subClassOf geop:Implementation ;
    rdfs:comment "The implementation of a geoprocessing operation as it is implemented
in ArcMap"^^xsd:string ;
    rdfs:label "ArcMap Implementation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#ClipOperation
geop:ClipOperation rdf:type owl:Class ;
    rdfs:subClassOf geop:RasterOperation ;
    rdfs:comment "Cuts out a portion of a raster dataset, mosaic dataset, or image service
layer"^^xsd:string ;
    rdfs:label "Clip Operation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#GeoOperation
geop:GeoOperation rdf:type owl:Class ;
    rdfs:comment "A typical geoprocessing operation takes an input dataset, performs an
operation on that dataset, and returns the result of the operation as an output dataset."^^xsd:string
;
    rdfs:label "Geoprocessing operation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#ILWISImplementation
geop:ILWISImplementation rdf:type owl:Class ;
    rdfs:subClassOf geop:Implementation ;
    rdfs:comment "The implementation of a geoprocessing operation as it is implemented
in ILWIS"^^xsd:string ;
    rdfs:label "ILWIS implementation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#Implementation
geop:Implementation rdf:type owl:Class ;
    rdfs:comment "The implementation of a geoprocessing operation in a specific GIS
client"^^xsd:string ;
    rdfs:label "Implementation"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#Input
geop:Input rdf:type owl:Class ;
    rdfs:subClassOf geop:Parameter ;
    owl:disjointWith geop:Output ;
    rdfs:comment "A sub-class of parameter describing the input parameters of geoprocessing
operations"^^xsd:string ;
    rdfs:label "Input Parameter"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#Output
geop:Output rdf:type owl:Class ;
    rdfs:subClassOf geop:Parameter ;
    rdfs:comment "A sub-class of parameter describing the output parameters of geoprocessing
operations"^^xsd:string ;
    rdfs:label "Output Parameter"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#Parameter
geop:Parameter rdf:type owl:Class ;
    rdfs:comment "A class describing the input and output parameters for geoprocessing
operations"^^xsd:string ;
    rdfs:label "Parameter"^^xsd:string .
```

```

### http://www.semanticweb.org/ontology/GeoOperations#PolygonToRaster
geop:PolygonToRaster rdf:type owl:Class ;
    rdfs:subClassOf geop:VectorOperation ;
    rdfs:comment "Converts polygon features to a raster dataset"^^xsd:string ;
    rdfs:label "Polygon To Raster"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#ProjectRaster
geop:ProjectRaster rdf:type owl:Class ;
    rdfs:subClassOf geop:RasterOperation ;
    rdfs:comment "Transforms the raster dataset from one projection to another"^^xsd:string
;
    rdfs:label "Project Raster"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#RasterCalculation
geop:RasterCalculation rdf:type owl:Class ;
    rdfs:subClassOf geop:RasterOperation ;
    rdfs:comment "Builds and executes a single Map Algebra expression"^^xsd:string ;
    rdfs:label "Raster Calculation"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#RasterOperation
geop:RasterOperation rdf:type owl:Class ;
    rdfs:subClassOf geop:GeoOperation ;
    rdfs:comment "A geoprocessing operation performed on a raster dataset."^^xsd:string ;
    rdfs:label "Raster Operation"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#ResampleOperation
geop:ResampleOperation rdf:type owl:Class ;
    rdfs:subClassOf geop:RasterOperation ;
    rdfs:comment "a tool used to change the spatial resolution of a raster
dataset"^^xsd:string ;
    rdfs:label "Resample Operation"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#VectorOperation
geop:VectorOperation rdf:type owl:Class ;
    rdfs:subClassOf geop:GeoOperation ;
    rdfs:comment "A geoprocessing operation performed on a vector dataset."^^xsd:string ;
    rdfs:label "Vector Operation"^^xsd:string .

# Individuals

### http://www.semanticweb.org/ontology/GeoOperations#cellSize
geop:cellSize rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Specify the cell size of the new raster"^^xsd:string ;
    rdfs:label "Cell Size"^^xsd:string .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#clip_am
geop:clip_am rdf:type owl:NamedIndividual ,
    geop:ArcMapImplementation ,
    geop:ClipOperation ;
geop:hasInput geop:inputRaster ,
    geop:rectangle ;
geop:hasOutput geop:outputRaster ;
geop:asPython """ClipRasters = arcpy.ListRasters(ClipOperationInputRaster + '*', \All\")
for i in ClipRasters:
    arcpy.Clip_management(i, "", ClipOperationOutputRaster + str(ClipRasters.index(i)),
ClipOperationRectangle)
print \Clip operation Successful\""""^^xsd:string ;
geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-
toolbox/clip.htm"^^xsd:anyURI ;
geop:software "ArcMap"^^xsd:string .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#clip_il
geop:clip_il rdf:type owl:NamedIndividual ,
    geop:ClipOperation ,
    geop:ILWISImplementation ;
geop:hasInput geop:inputRaster ,
    geop:rectangle ;
geop:hasOutput geop:outputRaster ;
geop:asPython """exec(\inputlist =\ + ClipOperationInputRaster)
exec(\rectangle =\ + ClipOperationRectangle)
clipList = []
for i in inputlist:
    result = Engine.do('mapcalc', 'iff(@1>=0,@2,0)',rectangle,i)
    clipList.append(result)
exec(ClipOperationOutputRaster + \ = clipList\")
print('Clip succesful')""""^^xsd:string ;
geop:documentation "https://github.com/52North/ILWISCore/wiki/Python-API-Clip-a-
Raster"^^xsd:anyURI ;
geop:software "ILWIS"^^xsd:string ;
rdfs:comment "generates a new boolean map based on the logical condition" .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#expression
geop:expression rdf:type owl:NamedIndividual ,
    geop:Input ;
geop:optional "false"^^xsd:boolean ;
rdfs:comment "Specify the input rasters for the raster calculation"^^xsd:string ;
rdfs:label "Expression"^^xsd:string .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#georeference
geop:georeference rdf:type owl:NamedIndividual ,
    geop:Input ;
rdfs:comment "Specify the file name including extension of a raster with the preferred
projection and cell size, or a georeference file containing this information." ;

```

rdfs:label "Georeference" .

```
### http://www.semanticweb.org/ontology/GeoOperations#inputFeatures
geop:inputFeatures rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Specify a vector dataset including extension" ;
    rdfs:label "Input Features"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#inputGeoreference
geop:inputGeoreference rdf:type owl:NamedIndividual ,
    geop:Input ;
    rdfs:comment "Specify the geographic projection of the input dataset by EPSG code.
See (see: http://spatialreference.org/ref/epsg/)"^^xsd:string ;
    rdfs:label "Input Georeference"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#inputRaster
geop:inputRaster rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment ""Specify the input raster for this dataset, unless a value is already present
in this input field.
```

```
If this is the first input raster of the workflow specify a unique section of the filename without
extension. (e.g. Filename: AfricaRainfall012016, input: Africa)."^^xsd:string ;
    rdfs:label "Input Raster"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#outputCoordinateSystem
geop:outputCoordinateSystem rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Input the EPSG code for the coordinate system the dataset needs to
be projected to (see: http://spatialreference.org/ref/epsg/)"^^xsd:string ;
    rdfs:label "Output Coordinate System"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#outputRaster
geop:outputRaster rdf:type owl:NamedIndividual ,
    geop:Output ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Specify the name of the raster you are creating, no extension
needed"^^xsd:string ;
    rdfs:label "Output Raster"^^xsd:string .
```

```
### http://www.semanticweb.org/ontology/GeoOperations#poltoras_am
geop:poltoras_am rdf:type owl:NamedIndividual ,
    geop:ArcMapImplementation ,
    geop:PolygonToRaster ;
    geop:hasInput geop:cellSize ,
```

```

        geop:inputFeatures ,
        geop:valueField ;
    geop:hasOutput geop:outputRaster ;
    geop:asPython
"""arcpy.PolygonToRaster_conversion(PolygonToRasterInputFeatures,PolygonToRasterValueField,Po
lygonToRasterOutputRaster, " , ", int(PolygonToRasterCellSize))

print \"Polygon to Raster succesful\""""^^xsd:string ;
    geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/conversion-
toolbox/polygon-to-raster.htm"^^xsd:anyURI ;
    geop:software "ArcMap"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#poltoras_il
geop:poltoras_il rdf:type owl:NamedIndividual ,
    geop:ILWISImplementation ,
    geop:PolygonToRaster ;
    geop:hasInput geop:georeference ,
        geop:inputFeatures ;
    geop:hasOutput geop:outputRaster ;
    geop:asPython """georef = createGeoreference(PolygonToRasterGeoreference)
fc = FeatureCoverage(PolygonToRasterInputFeatures)
exec(PolygonToRasterOutputRaster + \" = Engine.do('polygon2raster',fc,georef)\")
print('Pol2Ras succesful')""" ;
    geop:documentation "Software still in development, no documentation available yet." ;
    geop:software "ILWIS"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#proras_am
geop:proras_am rdf:type owl:NamedIndividual ,
    geop:ArcMapImplementation ,
    geop:ProjectRaster ;
    geop:hasInput geop:cellSize ,
        geop:inputGeoreference ,
        geop:inputRaster ,
        geop:outputCoordinateSystem ;
    geop:hasOutput geop:outputRaster ;
    geop:asPython """ProjectRasters = arcpy.ListRasters(ProjectRasterInputRaster + '*', \"ASC\")
outputcoord = arcpy.SpatialReference(int(ProjectRasterOutputCoordinateSystem))
inputcoord = arcpy.SpatialReference(int(ProjectRasterInputGeoreference))

for i in ProjectRasters:
    arcpy.ProjectRaster_management(i, ProjectRasterOutputRaster +str(ProjectRasters.index(i)),
outputcoord, \"Bilinear\", int(ProjectRasterCellSize), " , ", inputcoord)
print \"Project Raster Succesful\""""^^xsd:string ;
    geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/data-
management-toolbox/project-raster.htm"^^xsd:anyURI ;
    geop:software "ArcMap"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#proras_il

```

```

geop:proras_il rdf:type owl:NamedIndividual ,
    geop:ILWISImplementation ,
    geop:ProjectRaster ;
geop:hasInput geop:georeference ,
    geop:inputRaster ;
geop:hasOutput geop:outputRaster ;
geop:asPython """georef = createGeoreference(ProjectRasterGeoreference)
resampleList = []
exec("\inputlist =\ " + ProjectRasterInputRaster)
for i in inputlist:
    rcResample = Engine.do("\resample\ ", i, georef, "\bilinear\ ")
    resampleList.append(rcResample)
exec(ProjectRasterOutputRaster + "\ = resampleList\ ")
print('reproject succesful')""";
geop:documentation "https://github.com/52North/ILWISCore/wiki/Python-API-Reproject-
Raster-Coverages"^^xsd:anyURI ;
geop:software "ILWIS"^^xsd:string .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#rascal_am
geop:rascal_am rdf:type owl:NamedIndividual ,
    geop:ArcMapImplementation ,
    geop:RasterCalculation ;
geop:hasInput geop:expression ;
geop:hasOutput geop:outputRaster ;
geop:asPython """SumRasters = arcpy.ListRasters(RasterCalculationExpression + '*', \All\ ")
i = 1
sum = SumRasters[0]
while i < len(SumRasters):
    sum += Raster(SumRasters[i])
    i += 1
sum.save(RasterCalculationOutputRaster)
print "\Workflow succesfully executed\ """"^^xsd:string ;
geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/spatial-analyst-
toolbox/raster-calculator.htm"^^xsd:anyURI ;
geop:software "ArcMap"^^xsd:string .

```

```

### http://www.semanticweb.org/ontology/GeoOperations#rascal_il
geop:rascal_il rdf:type owl:NamedIndividual ,
    geop:ILWISImplementation ,
    geop:RasterCalculation ;
geop:hasInput geop:expression ;
geop:hasOutput geop:outputRaster ;
geop:asPython """exec("\inputlist = \ " + RasterCalculationExpression)
i = 1
sum = inputlist[0]
while i < len(inputlist):
    sum += inputlist[i]
    i += 1

```

```

sum.store(RasterCalculationOutputRaster + "\".mpr\", \"map\", \"ILWIS3\")\"";
    geop:documentation "Software still in development, documentation not yet available." ;
    geop:software "ILWIS"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#rectangle
geop:rectangle rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Input a raster datasets of the extent that needs to be clipped from the first
input dataset"^^xsd:string ;
    rdfs:label "Rectangle"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#res_am
geop:res_am rdf:type owl:NamedIndividual ,
    geop:ArcMapImplementation ,
    geop:ResampleOperation ;
    geop:hasInput geop:cellSize ,
    geop:inputRaster ;
    geop:hasOutput geop:outputRaster ;
    geop:asPython "Placeholder"^^xsd:string ;
    geop:documentation "http://desktop.arcgis.com/en/arcmap/latest/tools/data-management-
toolbox/resample.htm"^^xsd:anyURI ;
    geop:software "ArcMap"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#res_il
geop:res_il rdf:type owl:NamedIndividual ,
    geop:ILWISImplementation ,
    geop:ResampleOperation ;
    geop:hasInput geop:georeference ,
    geop:inputRaster ,
    geop:resamplingMethod ;
    geop:hasOutput geop:outputRaster ;
    geop:asPython "" ;
    geop:documentation "http://ILWIS.com" ;
    geop:software "ILWIS"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#resamplingMethod
geop:resamplingMethod rdf:type owl:NamedIndividual ,
    geop:Input ;
    rdfs:comment "The method used to aggregate the geometry of the output map, see
documentation to make a choice"^^xsd:string ;
    rdfs:label "Resampling Method"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#secondInputRaster
geop:secondInputRaster rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "Second input raster for comparison operation with first

```



```
raster^^xsd:string ;
    rdfs:label "Second Input Raster"^^xsd:string .

### http://www.semanticweb.org/ontology/GeoOperations#valueField
geop:valueField rdf:type owl:NamedIndividual ,
    geop:Input ;
    geop:optional "false"^^xsd:boolean ;
    rdfs:comment "The field in the attributes table used to assign values to the output
raster"^^xsd:string ;
    rdfs:label "Value Field"^^xsd:string .
```

## Appendix 2: Abstract workflow

```
{
  "workflows": [{
    "id": 0,
    "metadata": {
      "inputParameterCount": 4,
      "outputParameterCount": 2
    },
    "operations": [{
      "id": 0,
      "metadata": {
        "longName": "ProjectRaster",
        "inputParameterCount": 2,
        "outputParameterCount": 1,
        "final": false
      },
      "inputs": [{
        "id": 0,
        "name": "InputRaster"
      },
      {
        "id": 1,
        "name": "InterpolationMethod"
      }
    ],
      "outputs": [{
        "id": 0,
        "name": "OutputRaster"
      }
    ]
  },
  {
    "id": 1,
    "metadata": {
      "longName": "PolygonToRaster",
      "inputParameterCount": 2,
      "outputParameterCount": 1,
      "final": false
    },
    "inputs": [{
      "id": 0,
      "name": "InputFeature"
    },
    {
      "id": 1,
      "name": "CellSize"
    }
  ],
}
```

```

        "outputs": [{
            "id": 0,
            "name": "OutputRaster"
        }
    ],
    {
        "id": 2,
        "metadata": {
            "longName": "ClipOperation",
            "inputParameterCount": 2,
            "outputParameterCount": 1,
            "final": false
        },
        "inputs": [{
            "id": 0,
            "name": "InputRaster"
        },
        {
            "id": 1,
            "name": "Rectangle"
        }
    ],
        "outputs": [{
            "id": 0,
            "name": "OutputRaster"
        }
    ],
    {
        "id": 3,
        "metadata": {
            "longName": "RasterCalculation",
            "inputParameterCount": 1,
            "outputParameterCount": 1,
            "final": true
        },
        "inputs": [{
            "id": 0,
            "name": "Expression"
        }
    ],
        "outputs": [{
            "id": 0,
            "name": "OutputRaster"
        }
    ]
},
"connections": [{
    "fromOperationId": 0,
    "fromOutputId": 0,

```

```
        "toOperationId": 2,  
        "toParameterId": 0  
    },  
    {  
        "fromOperationId": 1,  
        "fromOutputId": 0,  
        "toOperationId": 2,  
        "toParameterId": 1  
    },  
    {  
        "fromOperationId": 2,  
        "fromOutputId": 0,  
        "toOperationId": 3,  
        "toParameterId": 0  
    }  
] ] }  
}
```

### Appendix 3: Executable Workflow ILWIS

```
#Declaration of the variables based on user input
workingDirectory = "d:/thesis/data/FullTestData/partialTest"
ProjectRasterInputRaster = "africa"
ProjectRasterGeoreference = "georefSource.tif"
PolygonToRasterOutputRaster = "outputBound"
PolygonToRasterGeoreference = "georefSource.tif"
PolygonToRasterInputFeatures = "boundaries_mamase.shp"
ClipOperationInputRaster = "projectOutput"
ProjectRasterOutputRaster = "projectOutput"
RasterCalculationExpression = "outputClip"
ClipOperationRectangle = "outputBound"
RasterCalculationOutputRaster = "finalResult"
ClipOperationOutputRaster = "outputClip"

import os
import fnmatch
from ILWIS import *

#Function that creates a list from the rasters so that a process can easily be performed on all rasters
def CreateRasterList (workingDirectory, filename):
    rasterlist = []
    list = os.listdir(workingDirectory)
    for file in list:
        if fnmatch.fnmatch(file, '*' + filename + '*.asc'):
            rascov = RasterCoverage(file)
            rasterlist.append(rascov)
    return rasterlist

#Function that allows the creation of a georeference from an input raster
def createGeoreference (georef):
    rc = RasterCoverage(georef)
    georeference=rc.geoReference()
    return georeference

#set ILWIS working directory
workingCatalog = 'file:/// + workingDirectory
Engine.setWorkingCatalog(workingCatalog)
exec(ProjectRasterInputRaster + '= CreateRasterList (workingDirectory, ProjectRasterInputRaster)')
print(africa)

georef = createGeoreference(ProjectRasterGeoreference)
resampleList = []
exec("inputlist =" + ProjectRasterInputRaster)
print(inputlist)
for i in inputlist:
    rcResample = Engine.do("resample", i, georef, "bilinear")
    resampleList.append(rcResample)
```

```

    print("One raster reprojected")
exec(ProjectRasterOutputRaster + "= resampleList")
print('reproject succesful')

georef = createGeoreference(PolygonToRasterGeoreference)
fc = FeatureCoverage(PolygonToRasterInputFeatures)
exec(PolygonToRasterOutputRaster + " = Engine.do('polygon2raster',fc,georef)")
print('Pol2Ras succesful')

exec("inputlist =" + ClipOperationInputRaster)
exec("rectangle =" + ClipOperationRectangle)
clipList = []
print(inputlist)
for i in inputlist:
    result = Engine.do('mapcalc','iff(@1>=0,@2,0)',rectangle,i)
    clipList.append(result)
exec(ClipOperationOutputRaster + " = clipList")
print('Clip succesful')

exec("inputlist =" + RasterCalculationExpression)
print(inputlist)
i = 1
sum = inputlist[0]
while i < len(inputlist):
    sum += inputlist[i]
    i += 1
    print("Added Successfully")
sum.store("finalFull2.mpr", "map", "ILWIS3")

```

## Appendix 4: Executable workflow ArcMap

```
#Declaration of the variables based on user input
workingDirectory = "d:/Thesis/Data/FullTestData/ArcMapWorkflowOutput/"
ProjectRasterCellSize = "1000"
ProjectRasterOutputRaster = "projOut"
ProjectRasterInputRaster = "chirps"
ProjectRasterInputGeoreference = "4326"
ProjectRasterOutputCoordinateSystem = "32736"
PolygonToRasterCellSize = "1000"
PolygonToRasterOutputRaster = "outputBound"
PolygonToRasterInputFeatures = "boundaries_mamase.shp"
PolygonToRasterValueField = "FID"
ClipOperationInputRaster = "projOut"
ClipOperationOutputRaster = "outputClip"
ClipOperationRectangle = "outputBound"
RasterCalculationExpression = "outputClip"
RasterCalculationOutputRaster = "thesisklaar"

import arcpy
from arcpy.sa import *
arcpy.CheckOutExtension("Spatial")
arcpy.env.workspace = workingDirectory

arcpy.PolygonToRaster_conversion(PolygonToRasterInputFeatures,PolygonToRasterValueField,Polyg
onToRasterOutputRaster, "", "", int(PolygonToRasterCellSize))

print "Polygon to Raster succesful"
ProjectRasters = arcpy.ListRasters(ProjectRasterInputRaster + '*', "ASC")
outputcoord = arcpy.SpatialReference(int(ProjectRasterOutputCoordinateSystem))
inputcoord = arcpy.SpatialReference(int(ProjectRasterInputGeoreference))

for i in ProjectRasters:
    arcpy.ProjectRaster_management(i, ProjectRasterOutputRaster +str(ProjectRasters.index(i)),
outputcoord, "Bilinear", int(ProjectRasterCellSize), "", "", inputcoord)
print "Project Raster Succesful"

ClipRasters = arcpy.ListRasters(ClipOperationInputRaster + '*', "All")
for i in ClipRasters:
    arcpy.Clip_management(i, "", ClipOperationOutputRaster + str(ClipRasters.index(i)),
ClipOperationRectangle)
print "Clip operation Succesful"

SumRasters = arcpy.ListRasters(RasterCalculationExpression + '*', "All")
i = 1
sum = SumRasters[0]
while i < len(SumRasters):
    sum += Raster(SumRasters[i])
    i += 1
```

```
sum.save(RasterCalculationOutputRaster)  
print "Workflow succesfully executed"
```



## Appendix 5: User Manual for proof of concept application

This document will provide all information required to install the Spatial Problem-solving application. First a section will describe which tools and software needs to be installed, then a step by step user guide will be provided which describes how to use the application, and how to come to the results as they were presented in the thesis.

### Software required

In order to fully use the application there are quite some software requirements. These can be divided into two categories. Tools needed for running the application and creating the executable workflow, and tools needed for the execution of the output workflow.

#### Tools for running the application

- Install Node.js which can be downloaded here: <https://nodejs.org/en/>
- Install a triple store, to store and query data, using GraphDB would be easiest as the application is set up to use this triple store:  
<https://ontotext.com/products/graphdb/#button>

#### Tools for execution of executable workflow

##### Execution in ArcMap:

- Install Python 2.7.x: <https://www.python.org/downloads/>
- Install and obtain license for ArcMap: <https://www.esri.com/en-us/store/arcgis-desktop>

##### Execution in ILWIS:

- Install Python 3.5.x: <https://www.python.org/downloads/>
- Install ILWIS Object as provided with the application in the Software folder, this installer only works on Windows, using Python 3.5 32-bit.

### Preparing the triple store

This section describes how to prepare the GraphDB triple store to be used with this application. If you decide to use a different triple store, please edit the SPARQL endpoint specified in sparql.js in the Application folder to point to the endpoint of your selected triple store.

After downloading and installing GraphDB, open the GraphDB Workbench. By default, at <http://localhost:7200>. In the on the left side of the screen go to Setup and click Repositories (Figure 1). Here, you will set up a repository to store the ontology and operations instances in. On this page click the 'Create new repository' button and as Repository ID enter 'Thesis'. Leave all other settings as they are and click 'Create'.

Now go to the 'Import' tab in the menu (Figure 1) and select RDF. On this page click on the upload button (Figure 2). Navigate to the Ontology folder and select 'Geo-operation ontology.owl', and click 'Open'.

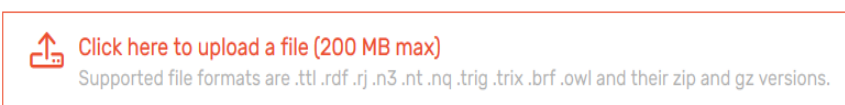


Figure 2 GraphDB upload button

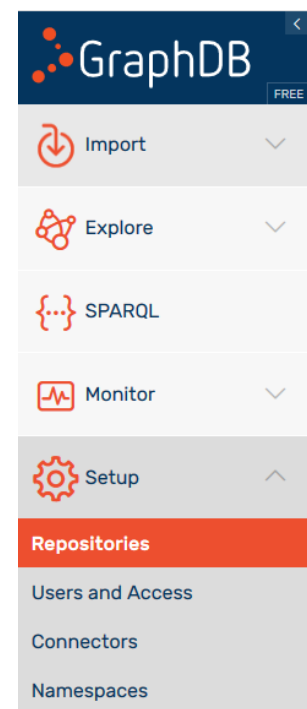


Figure 1 GraphDB menu

You will now see the geo-operation ontology show up in the list on your screen (Figure 3). Select it and click the import button, in the pop-up screen don't change any settings,

just click 'Import' again. The import should be successful, and the data is now accessible through the application.

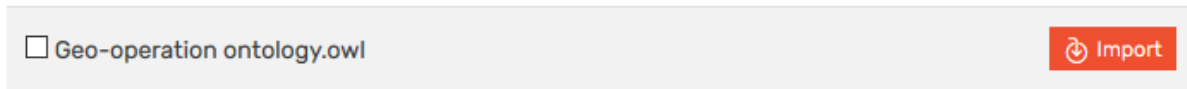


Figure 3 File list with import button

### Using the application

Now that the triple store is set up we can run the application. For this make sure Node.js is installed and working (see installation instructions on node.js website). Open command prompt in windows and navigate to the Application folder. Type the following command: 'Node app.js'. If it is working it should return: 'Listening on port 8888'.

Now in your browser go to <http://localhost:8888/upload>. You will see the workflow upload and software selection screen (Figure 4).

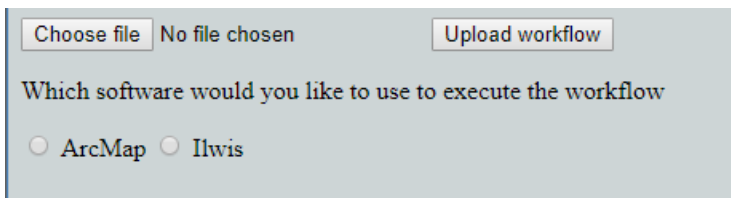


Figure 4 Workflow upload and software selection screen

Click the 'Choose file' button and navigate to the Workflows folder and select 'AbstractWorkflow.json'. Now please select the software you want to use for the execution of the workflow by clicking one of the radio buttons. Then proceed by clicking 'Upload workflow'. In the next screen, you will be provided with an overview of the operations used in the workflow and with links to the official documentation. Furthermore, the parameters for each operation are presented, together with an 'additional info' button and an input field. To get more information on the required input for a specific parameter click the info button behind the input field.

At the top of the page is an input field to specify the working directory. Please specify the path to the Data folder.

If you selected ArcMap as the execution software, see Figure 5 for the required user inputs to obtain the same results as are presented in the thesis.

Input	Inputfield	Info	Output	Outputfield	Info
Cell Size	1000		Output Raster	projOut	
Input Raster	chirps				
Input Georeference	4326				
Output Coordinate System	32736				
Cell Size	1000		Output Raster	outputBound	
Input Features	boundaries_mamase.shp				
Value Field	FID				
Input Raster	projOut		Output Raster	outputClip	
Rectangle	outputBound				
Expression	outputClip		Output Raster	rainRaster	

Figure 5 ArcMap inputs as used in application testing

If you selected ILWIS as the selected execution software, see Figure 6 for the required user inputs to obtain the same results as are presented in the thesis.












Input	Inputfield	Info	Output	Outputfield	Info
Input Raster	africa		Output Raster	outputRaster	
Georeference	georefSource.tif				
Input Features	boundaries_mamase.shp		Output Raster	outputBound	
Georeference	georefSource.tif				
Input Raster	outputRaster		Output Raster	outputClip	
Rectangle	outputBound				
Expression	outputClip		Output Raster	finalResult	

Figure 6 ILWIS inputs as used in application testing

Please feel free to change user inputs to your liking, but please note that ArcMap output raster names cannot be longer than 11 characters. However, when specifying the name of the output raster, this name will automatically be copied to the connected input field, please don't change this to retain the workflow logic.

When finished, please click the submit button. You will be prompted with a download screen where you can select the location to store the executable workflow.

#### Executing the workflow

To execute the workflow please install the required software as specified at the start of this manual. If ArcMap was selected as execution software, make sure ArcMap is installed, and open the executable workflow in a Python 2.7 editor. From here, run the workflow. Execution of the workflow may take several hours, depending on the hardware you are using. When execution is finished, the final result can be found in the Data folder under the name specified in the application as output of the Raster Calculation. It can be opened in a GIS of your choosing, for inspection.

If ILWIS was selected as execution software, make sure ILWIS Objects is installed (specifically the version present in the Software folder). Furthermore, make sure Python 3.5 is installed. Open the executable workflow in a Python 3.5 editor, and from here, run the workflow. Execution takes considerably longer than in ArcMap and may take several hours, also depending on your hardware. When execution is finished it, the final result can be found in the Data folder, under the name specified in the application as output of the Raster Calculation. It will be stored as a .mpr file, and can be opened in a limited range of GIS, including ILWIS and QGIS, for inspection.