



D4.6 Release of Open Source Framework V2.0

<i>File name</i>	PuppyIR-D4.6-Release-of-Open-Source-Framework-V2.0
<i>Author(s)</i>	Richard Glassey (UGLW) Douglas Dowie (UGLW) Leif Azzopardi (UGLW) José Miguel Garrido (ATOS) Jeldrik Schmuch (ATOS) Paul Moore (ATOS)
<i>Work package/task</i>	WP4
<i>Document status</i>	Draft
<i>Version</i>	Final
<i>Contractual delivery date</i>	M30
<i>Confidentiality</i>	Confidential
<i>Keywords</i>	Framework Implementation and Release
<i>Abstract</i>	Release notes to accompany the release of the second major version of the PuppyIR Framework.

Table of Contents

Executive Summary.....	2
1 Introduction.....	3
1.1 Audience.....	3
1.2 Implications from D4.5 – Report on Specification and Design Changes.....	3
1.3 Report Structure.....	3
2 Framework Map.....	4
3 Framework Features.....	5
3.1 Search Engines.....	6
3.2 Query Filters and Modifiers.....	7
3.3 Query Loggers.....	7
3.4 Result Filters and Modifiers.....	8
4 Framework Documentation.....	9
5 Dependencies and Installation.....	12
5.1 Core Dependencies.....	12
5.2 Additional Requirements.....	12
5.3 Installation Notes.....	12
5.3.1 Install Python.....	13
5.3.2 Install Universal Feed Parser.....	13
5.3.3 Install lxml.....	13
5.3.4 Install BeautifulSoup.....	13
5.3.5 Install PuppyIR.....	13
5.3.6 Install Django.....	14
5.3.7 Install Python Imaging Library (PIL).....	14
5.3.8 Install Whoosh.....	14
6 Development Roadmap.....	15
6.1 Integration of PuppyIR and Django.....	15
6.2 Configuration of PuppyIR Services.....	15
7 Summary.....	18
8 References.....	19

Executive Summary

This report accompanies the second major release of the PuppyIR Framework. These release notes provide a map of the framework development activities, the main features of the framework, the requirements and installation guide, and an outline for the development activities ahead of the third version of the framework.

This deliverable builds upon the previous WP1, WP3 and WP4 deliverables:

- D1.2 – Agreed User Requirements and Scenarios **[2]**
- D1.3 – Agreed Technical Requirements **[3]**
- D3.1 – Report on Data Pre-processing **[4]**
- D4.1 – Specification Report **[5]**
- D4.2 – Design Report **[6]**
- D4.3 – Report on Implementation and Documentation **[7]**
- D4.4 – Release of the Open Source Framework V1.0 **[8]**
- D4.5 – Report on Design and Specification Changes **[9]**

The source code can be accessed from the project's SVN repository hosted on Sourceforge at:

<http://sourceforge.net/projects/puppyir/>.

The documentation can be accessed online at the following address:

<http://www.puppyir.eu/framework>

1 Introduction

The PuppyIR Project aims to facilitate the creation of child-centred information services, based on the understanding of the behaviour and needs of children [1]. The goals of this report are to:

- 1) Provide a map of the framework contents and status
- 2) Outline of the main features
- 3) Introduce the new documentation for the framework
- 4) Describe the dependencies and installation process
- 5) Provide a roadmap for the development of the second version

This report serves 3 key purposes:

1. Provide release notes to accompany the source code for the second version of the framework for project developers.
2. Describes the current framework features.
3. Communicate to all project members and stakeholders how the project will be progressing from the interim release towards the final version.

1.1 Audience

This deliverable is intended to be read by both members and developers of the PuppyIR project to communicate the current status of the framework development. It is also intended to be read by third party developers who are planning on using the framework.

1.2 Implications from D4.5 – Report on Specification and Design Changes

D4.5 [9] reported the changes made to the original specification and design reported in earlier deliverables, D4.1 – Specification Report [5] and D4.2 – Design Report [6]. These changes are reflected in the framework code that has evolved from the first release into the second release. Since D4.5 there have been no major design or specification changes that need to be reported and instead the framework code base has been mostly refactored to improve the comprehensibility and clarity for third party developers.

1.3 Report Structure

The report is divided into the following major sections:

- **Section 2 – Framework Map**
- **Section 3 – Framework Features**
- **Section 4 – Framework Documentation**
- **Section 5 – Dependencies and Installation**
- **Section 6 – Development Roadmap**

2 Framework Map

The growth of the PuppyIR framework is illustrated in Fig. 1, a tree diagram of the current state of the Sourceforge repository as of V2.0 of the framework. The bold paths highlight the main development branches, and the grey paths show the other main developments within the project that are now publically available on Sourceforge.

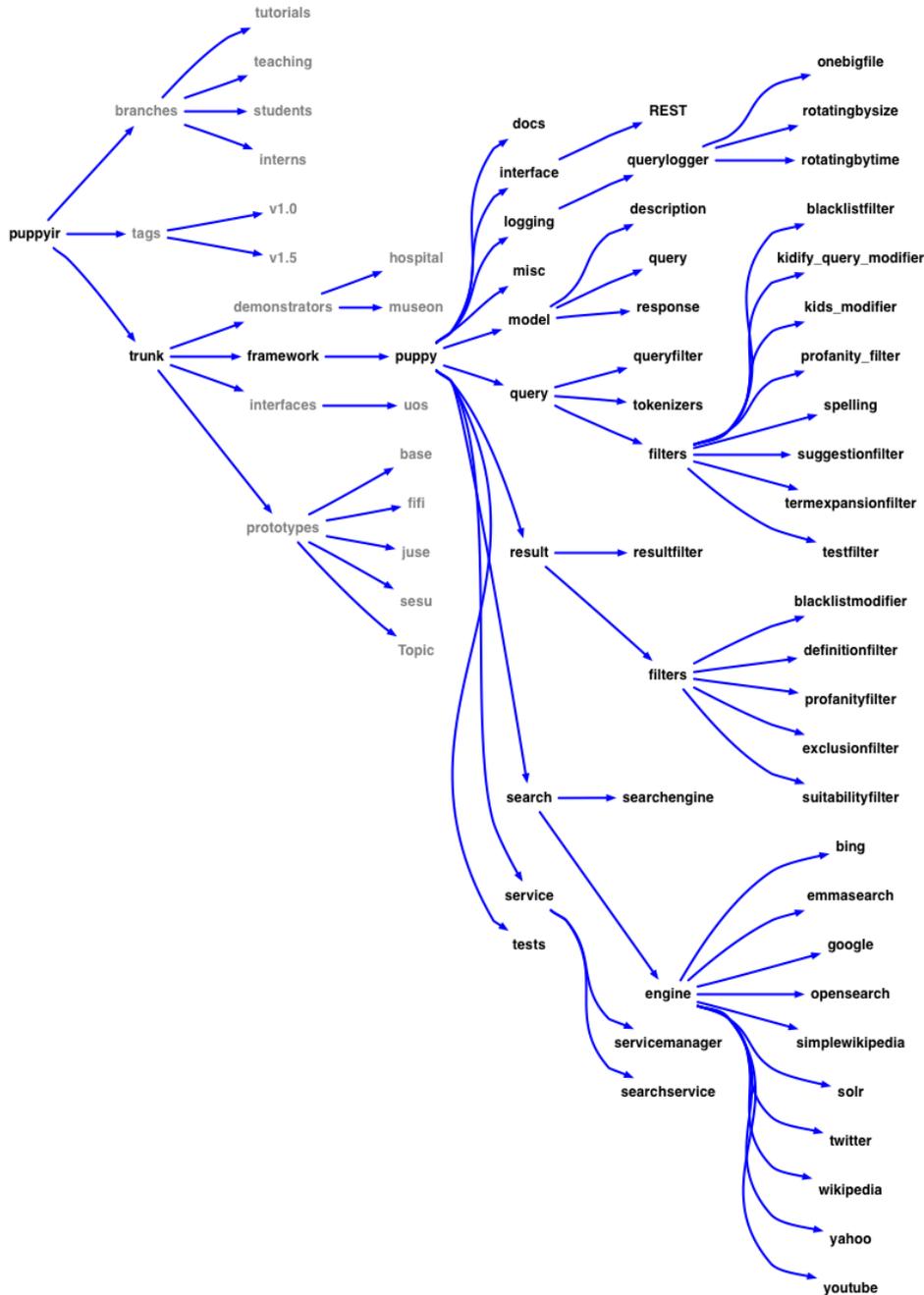


Figure 1: A map of the PuppyIR Sourceforge repository outlining the scope of the development

3 Framework Features

The current version of the framework supports the construction of custom search services, which take the form of a pipeline consisting of query filters, query modifiers, query loggers, search engines, result filters and result modifiers, as shown in the figure below.

Queries and Results both contain filters and modifiers (as noted above). The distinction is as follows:

- **Filters:** these reject or accept a query, or result, based on a defined criteria. For example a blacklist filter rejects queries containing one or more blacklisted words.
- **Modifiers:** these change the content of a query, or result, based on a defined behaviour. For example, appending “for kids” to every query.

Figures 2 and 3 show the Query and Result pipelines. Within these pipelines, all the filters are executed first and following this, the modifiers are executed. Filters and modifiers are stored in the framework under the “filter” category (in both query and result) as shown in Figure 1.

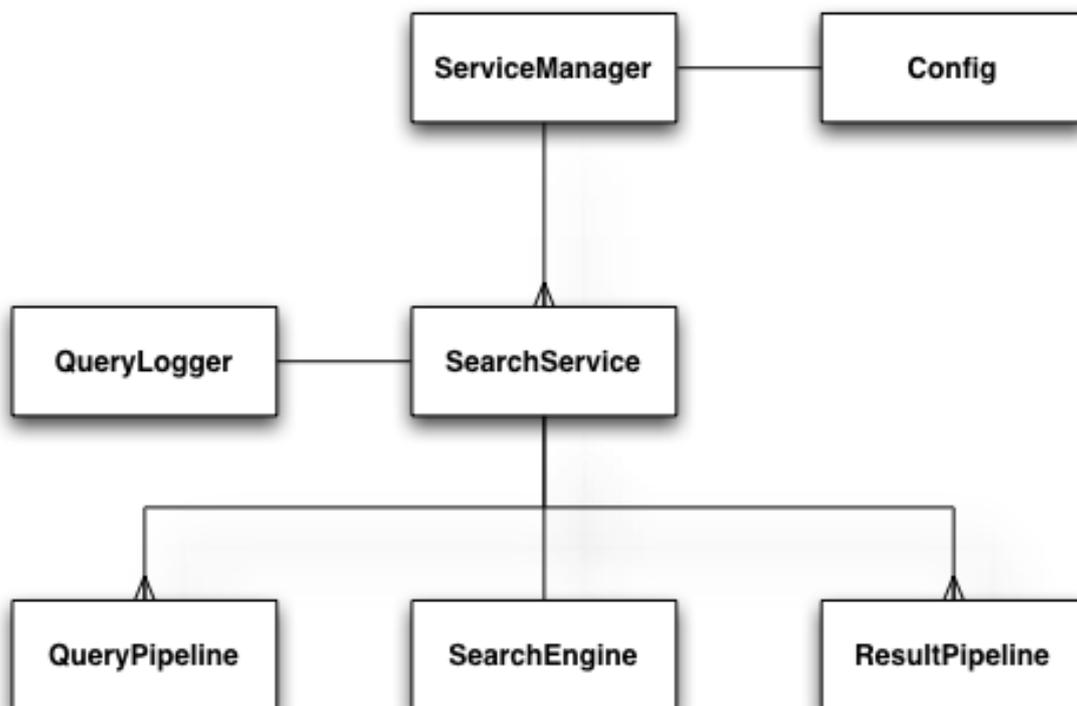


Figure 2: High level view of a PuppyIR service, illustrating the pipeline architecture.

The main flexibility and extensibility that the framework provides comes from this pipeline architecture. Developers can inherit and extend functionality in the key areas of query processing, search engine support and result processing. The framework ensures that queries and sets of results are translated into a common data model that all components can interpret and manipulate. Whilst single pipelines are useful, more complex services can be composed by combining multiple pipelines into a single service, with the most common example being an additional pipeline to manage search suggestions, as shown below.

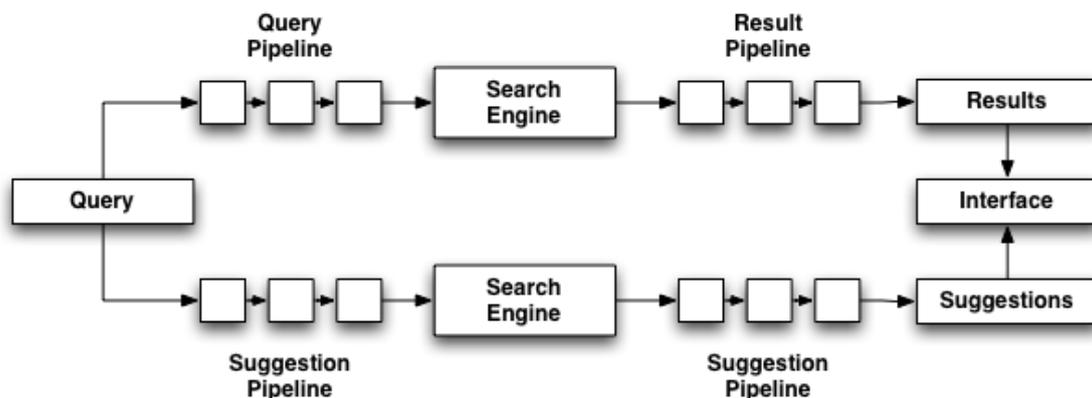


Figure 3: Service composed of multiple pipelines; one to conduct the main search and the other to provide search suggestions to assist in the search process.

The core framework already includes a number of ready made query filters, query modifiers, search engine wrappers, result filters and result modifiers to allow a developer to build a service. Over time it is hoped that more components are developed by third parties to create a richer component library for building services. The following components are available as of V2.0:

3.1 Search Engines

The following search engine wrappers have been implemented:

- **Bing**
 - The Microsoft Bing Search API
 - <http://www.bing.com/toolbox/bingdeveloper/>
- **Emmasearch**
 - An index of the Emma Kinderziekenhaus data that has been curated and indexed within the PuppyIR project as part of Work Package 3 and deliverable D3.1 – Report of Data Pre-processing [4]. The search service is built using PF/Tijah (<http://dbappl.cs.utwente.nl/pftijah/>)
- **Google**
 - Unfortunately this wrapper is now deprecated.
- **Opensearch**
 - This is a good example of a custom search wrapper. Rather than act as a simple wrapper to an existing search API (e.g. Yahoo!), this search wrapper takes a URL as an argument, then attempts to discover if the site at the URL is Opensearch compatible. If a standard description of how to search the site is available, then this wrapper will add the site as a search service.
 - See Opensearch Standard for more information: <http://www.opensearch.org/>
- **Wikipedia and SimpleWikipedia**
 - These are convenience wrappers for these popular information sites. Although both are powered by the Mediawiki content management system, which itself is Opensearch compliant, their popularity has earned them a dedicated wrapper.
 - Furthermore, the results returned often contain hyperlinks to thumbnails. With a little bit of link manipulation, the high quality image link can be returned instead.

- **Solr/Lucene and Whoosh**
 - Most search wrappers operate by parsing the results of an online search API. However there are instances when indexing content with an engine such as Solr/Lucene (Java) or Whoosh (Python) is a necessary requirement of a service.
- **Twitter**
 - Moving into the social sphere, services like Twitter expose search APIs and return results in the Opensearch format. This wrapper illustrates how trivial it is to add more novel source of information that may help improve a service (whilst also making the content moderation more challenging!)
- **Yahoo!**
 - At present, the Yahoo search backend uses Bing, however the result ranking does vary enough to warrant this wrapper.
- **Youtube**
 - Besides textual information, rich media results are a crucial source of information that is likely to stimulate interest for young searchers.

3.2 Query Filters and Modifiers

The following query filters and modifiers are available to construct search service pipelines:

- **Blacklist Filter**
 - Rejects any queries containing terms found in the blacklist.
- **Kids Modifier**
 - Adds “for kids” to the query terms, then checks against the search suggestions from the Google suggest utility to discover if this is a frequent query modification, otherwise returns the default query terms.
- **Profanity Filter**
 - Rejects any queries containing profanity (the WYDL service is used to detect profanity).
- **Suggestion Filter (behaves like a modifier)¹**
 - Adds a list of search suggestions to the original query metadata.
- **Term Expansion Filter (behaves like a modifier)**
 - Adds additional terms to the original query terms.

3.3 Query Loggers

Every service can be configured to log queries. As a service can be composed of multiple search services, individual log files for each service are created and maintained. The type of log file management can be controlled by choosing one of the following options:

- **OneBigFile**
 - Store all queries into one file. This mode simplifies the analysis of query data, but is not suitable for high volume services.

¹ As of July 2011, the web service used by this filter (to generate suggestions) no longer responds and so, only default suggestions are used.

- **RotateBySize**
 - Alternatively, a *maxBytes* parameter can be set to control the maximum size that a log file is allowed to grow to. Once this limit has been reached, a new file is created and the old file is archived and identified by a time-stamped filename.
- **RotateByTime**
 - Finally, a *time* parameter can be set if it is preferable to recycle the log files after a certain period of time (e.g. once a day or once a week). As before, old files are archived and named appropriately to identify their ordering.

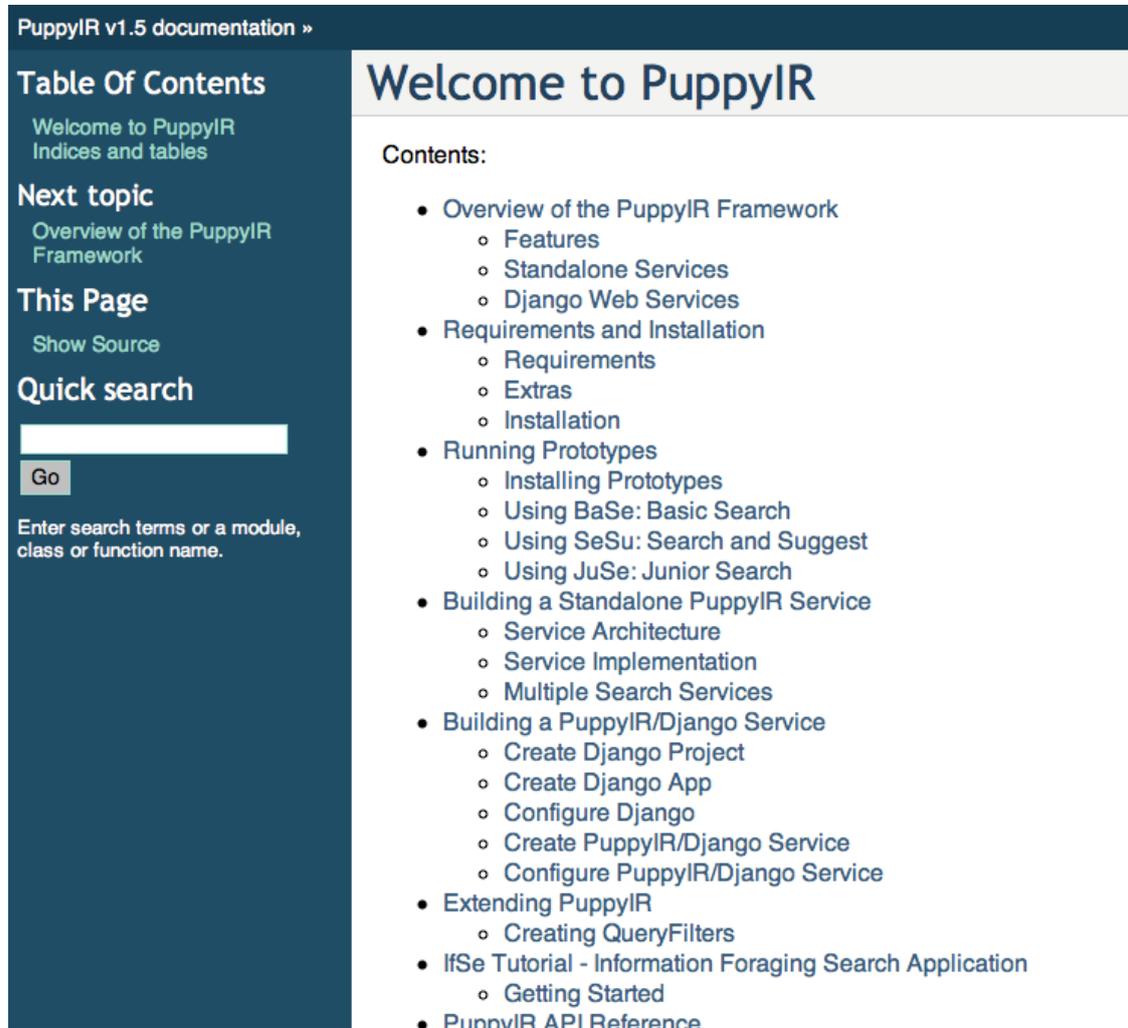
3.4 Result Filters and Modifiers

The following result filters and modifiers are available to construct search service pipelines:

- **Blacklist Modifier**
 - Any blacklisted terms found in a result, in the title or summary fields, will be removed by replacing the offending terms with `****`.
- **Exclusion Filter**
 - Presence of specific excluded sites will block results from being returned.
- **Suitability Filter**
 - Results are scored on their suitability (page readability, layout, graphics) for children and can be removed if a certain threshold is not met.
- **Profanity Filter**
 - Presence of profanity, in the title or summary fields, will cause the result to be rejected (the WYDL service is used to detect profanity).

4 Framework Documentation

In preparation for the v2.0 release, the framework documentation has been made available publically online. The welcome page and general index can be seen in Fig. 4.



The screenshot shows the 'Welcome to PuppyIR' page. On the left, there is a dark blue sidebar with navigation links: 'Table of Contents' (with sub-links for 'Welcome to PuppyIR' and 'Indices and tables'), 'Next topic' (with sub-link 'Overview of the PuppyIR Framework'), 'This Page' (with sub-link 'Show Source'), and 'Quick search' (with a search input field and a 'Go' button). Below the search field, it says 'Enter search terms or a module, class or function name.' The main content area has a light blue header 'Welcome to PuppyIR' and a 'Contents:' section with a bulleted list of topics and sub-topics.

PuppyIR v1.5 documentation »

Table of Contents

- Welcome to PuppyIR
- Indices and tables

Next topic

- Overview of the PuppyIR Framework

This Page

- Show Source

Quick search

Enter search terms or a module, class or function name.

Welcome to PuppyIR

Contents:

- Overview of the PuppyIR Framework
 - Features
 - Standalone Services
 - Django Web Services
- Requirements and Installation
 - Requirements
 - Extras
 - Installation
- Running Prototypes
 - Installing Prototypes
 - Using BaSe: Basic Search
 - Using SeSu: Search and Suggest
 - Using JuSe: Junior Search
- Building a Standalone PuppyIR Service
 - Service Architecture
 - Service Implementation
 - Multiple Search Services
- Building a PuppyIR/Django Service
 - Create Django Project
 - Create Django App
 - Configure Django
 - Create PuppyIR/Django Service
 - Configure PuppyIR/Django Service
- Extending PuppyIR
 - Creating QueryFilters
- IfSe Tutorial - Information Foraging Search Application
 - Getting Started
- PuppyIR API Reference

Figure 4: Screenshot of the welcome page and index to the official PuppyIR documentation, available at <http://www.puppyir.eu>

Beyond the basic API reference for the framework, there are sections devoted to the basics of installation and setup, how to run some of the prototypes that project members have developed to get a feel for the functionality of the framework, as well as detailed walkthroughs on how to starting coding with the framework.

Service Implementation

The following steps will create and configure a new service, adding a search engine, query logger, and query and result filtering.

1. Create and Configure the ServiceManager

Create a new python script, e.g. service.py

```
from puppy.service import ServiceManager

# Determine your local config options if any (e.g. api keys and proxies)
# e.g. the university network requires a proxy for all web traffic
config = {
    "proxyhost": "http://wwwcache.gla.ac.uk:8080",
}

# Create the ServiceManager
sm = ServiceManager(config)
```

2. Create a SearchService

```
# new imports
from puppy.service import ServiceManager, SearchService

config = {
    "proxyhost": "http://wwwcache.gla.ac.uk:8080",
}

sm = ServiceManager(config)

# create SearchService and give it a name
ss = SearchService(sm, "bing_web")
```

Figure 5: Code samples are provided to assist 3rd party developers in getting acquainted with the PuppyIR Framework

Figure 5 illustrates the detail that these guides provide, helping new developers rapidly adopt the framework and build services for themselves. We have taken the opportunity to test out this documentation with both students in a Web Engineering class, as well as an EU summer school for Information Foraging.

As the framework matures and more feedback on the documentation is gathered, improvements will be made for the final v3.0 release at the end of project.

Downloading the Source Code for the Tutorial

Download the latest release of the tutorial from the PuppyIR repository:

```
$ svn export https://puppyir.svn.sourceforge.net/svnroot/puppyir/branches/tutorials/ifse-tutorial
```

Run IfSe

```
$ cd /path/to/ifse-tutorial  
$ python manage.py runserver
```

Visit <http://127.0.0.1:8000/ifse> this should bring up interface below.

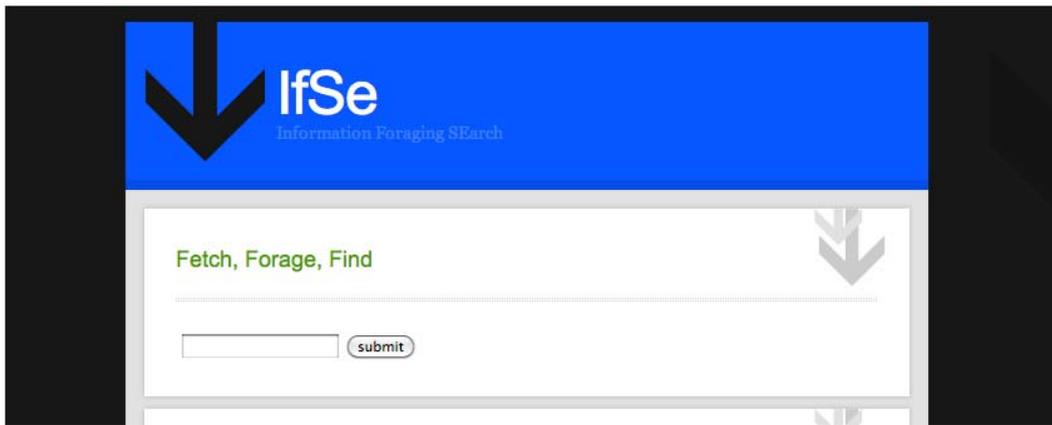


Figure 6: A full tutorial is included that allows developers to create a service within a few lines of code, and then explore the available components

A further addition to the documentation is a tutorial aimed at users who may have no programming experience whatsoever. This 'hands-free' tutorial allows non-technical users to experiment with the framework and build a functioning system with very little effort. A future interest here is attempting to allow children themselves to follow a similar guide and design their own search services, allowing researchers to take inspiration and guidance from the designs and requirements that children generate on their own.

5 Dependencies and Installation

The official guide for dependencies and installation is now hosted online at this address: <http://www.puppyir.eu/framework>. This will be updated at regular intervals in line with major releases. The most up to date documentation is always available from the Sourceforge repository, and can be accessed using the following command:

```
$ svn export  
https://puppyir.svn.sourceforge.net/svnroot/puppyir/trunk/framework/puppy/docs
```

The following sections outline the core dependencies, additional requirements and a simple guide for installation.

5.1 Core Dependencies

For the basic framework:

- Python Programming Environment <http://www.python.org/>
- Universal Feed Parser <http://www.feedparser.org/>
- lxml <http://pypi.python.org/pypi/lxml/>
- BeautifulSoup <http://www.crummy.com/software/BeautifulSoup/#Download>
- PuppyIR <http://sourceforge.net/projects/puppyir/>

5.2 Additional Requirements

To create web services:

- Django Web Application Framework <https://www.djangoproject.com/> (see section 6.1)

To run the prototype services in the PuppyIR framework:

- PIL <http://www.pythonware.com/products/pil/>

To include a full text indexer:

- Whoosh <http://pypi.python.org/pypi/Whoosh/#downloads>

5.3 Installation Notes

The following sections provide instructions on installing each of the requirements.

Note: If you are running MacOS X, please ensure that you have X-Code (<http://developer.apple.com/technologies/tools/>) installed. Either Version 3 or 4, this may be included on your install disc. This is required as several of the dependencies use X-Code's C compiler.

5.3.1 Install Python

If your system does not have Python installed, or you have an earlier version, you can find the latest 2.7 branch of Python here: <http://python.org/download/>. Follow the installation instructions for your own operating system.

At present, Python 3.0+ is not supported and may cause problems if installed. You can discover your current version of Python by launching a command prompt and typing the command 'python'. The version number should be displayed as shown below. If Python 3.0+ is installed, please install the earlier version (the 2.7 branch) to run PuppyIR.

```
$ python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

5.3.2 Install Universal Feed Parser

Download source from <http://www.feedparser.org/>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

5.3.3 Install lxml

Download the source from <http://pypi.python.org/pypi/lxml/>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

5.3.4 Install BeautifulSoup

Download the source from <http://www.crummy.com/software/BeautifulSoup/#Download>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

5.3.5 Install PuppyIR

Download the stable release from <http://sourceforge.net/projects/puppyir/files/release/>

```
$ cd path/to/puppyir
$ python setup.py install
```

Alternatively, the very latest release can be checked out of the repository:

```
$ svn export
https://puppyir.svn.sourceforge.net/svnroot/puppyir/trunk/framework puppyir
$ cd puppyir
$ python setup.py install
```

5.3.6 Install Django

(Only required if building web services)

Download source from <https://www.djangoproject.com/download/>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

5.3.7 Install Python Imaging Library (PIL)

(Only required for the JuSe prototype)

Download the source from <http://www.pythonware.com/products/pil/>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

5.3.8 Install Whoosh

(Only required for local full text indexing)

Note: Whoosh requires 'setuptools' to be installed before following the instructions below. This can be found at <http://pypi.python.org/pypi/setuptools>

Download the version compatible with the version of Python you are running. Follow the 'setuptools' installation instructions and then the steps detailed below.

Download source from <http://pypi.python.org/pypi/Whoosh/#downloads>

```
$ cd /path/to/source
$ python setup.py install # may require 'sudo'
```

6 Development Roadmap

The core components of the framework have stabilised, and will be refined further for the final release. Deliverable **D4.4 – Release of Open Source Framework V1.0** highlighted known issues and future developments which form the basis of the roadmap for the remaining development of the framework and additional components.

The first issue of **documentation** of the framework has been addressed, as discussed in section 4 of this deliverable. The second issue of **client interface** development is being addressed as part of work package 7, through the development of two web based interfaces: the hospital information service (see D7.3) and the post-visit museon information service (see D7.2), both of which are built using the framework.

The remaining issues that are under development are **integration** and **configuration** of PuppyIR services. Integration refers to the link between the framework and the Django web application framework. Configuration refers to the means by which a service can be altered to fit the requirements of its users. Each issue is discussed briefly in the following sections.

6.1 Integration of PuppyIR and Django

An early objective of the framework design was to keep it modular and independent of any third party web application framework. This has the benefit of allowing the framework to be explored in a *standalone mode*. This removes the burden of needing to also become familiar with a web application framework. Furthermore, the framework documentation has a section devoted to showing framework users how to build a complete search service that simply runs on the command line.

However, it is most probable that users of the framework will want to go beyond the command line to create an interesting service for children. This entails the development of a client interface and the hosting of a service so that it is web accessible. Throughout the project, developers have relied upon the Django web application framework to support these requirements. As this has been a successful combination in terms of *stability* (Django is a mature and well supported open source project) and *efficiency* (Django drastically speeds up the development of web applications), it has become the recommended method for third party developers to adopt.

Therefore, to reflect the integration of PuppyIR and Django a suite of basic setup utilities/scripts and documentation will be developed that achieve the following aims:

1. Create a basic Django web-application
2. Modify this web-application to become a PuppyIR service, providing a:
 - a. Skeleton PuppyIR service project layout
 - b. Default search service engine
 - c. Default search service interface
 - d. Default admin interface to configure search service

Ideally, this process would be encapsulated in several simple commands that a developer is required to execute. Afterwards, the developer should be able to visit a web page and use the default service without having written any code. It is foreseeable that this process could be repeated for any compatible web application framework, should a better alternative emerge.

6.2 Configuration of PuppyIR Services

Once a service is up and running, it may require configuration. The most desirable method of achieving this is through a web-based administrative interface. A further advantage of Django is the provision of a comprehensive web-based admin interface. For the purposes of PuppyIR, this

admin interface can be used to access a model of the service, manipulate it, and save new configurations without having to restart a service.

To illustrate this, a PuppyIR service can be modelled as the following set of relationships:

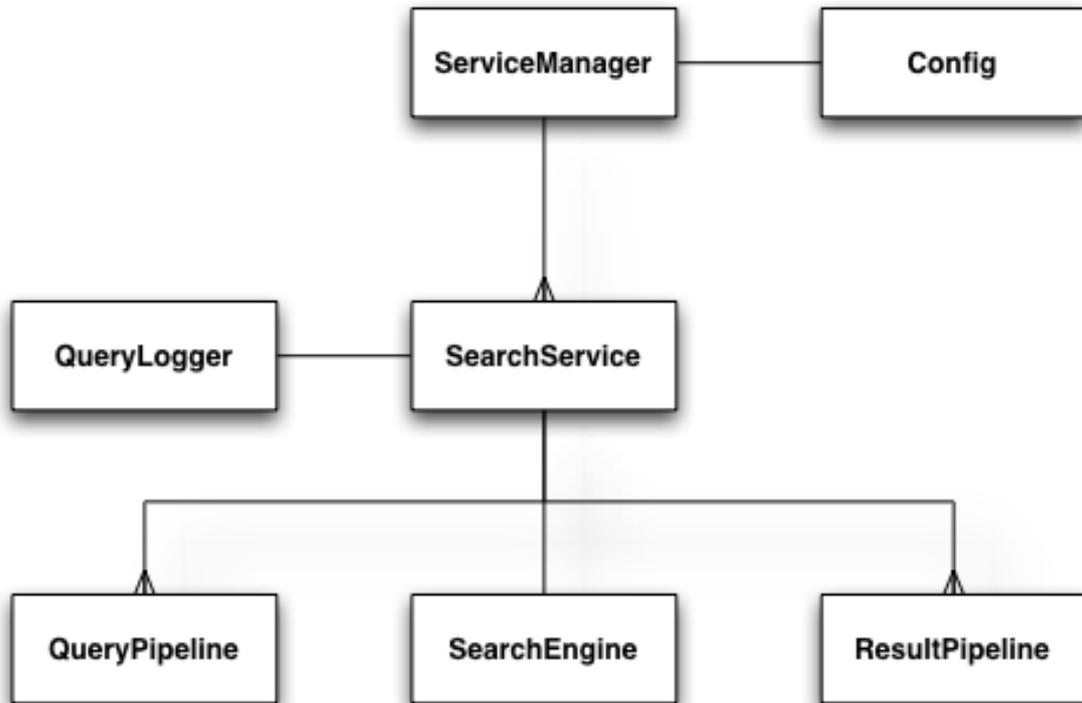


Figure 7: Abstract representation of a PuppyIR service model

This data model of a service can be modelled and stored in a database. The Django admin interface can then access and manipulate this model. The PuppyIR service can also refer to this model, and whenever it changes, it can reconfigure itself. The following screenshots illustrate an early version of this interface.

Pir	
Query filters	+ Add ✎ Change
Result filters	+ Add ✎ Change
Search engines	+ Add ✎ Change
Search services	+ Add ✎ Change
Service configs	+ Add ✎ Change

Figure 8: Django admin interface for customising a service

Django administration

Home > Pir > Search services > Add search service

Add search service

Name:

Searchengine: +

Query filters: +
Hold down "Control", or "Command" on a Mac, to select more than one.

Result filters: +
Hold down "Control", or "Command" on a Mac, to select more than one.

Figure 9: Django admin interface for adding a new search service, choosing a search engine, and adding multiple query and result filters

Django administration

Home > Pir > Query filters > simplefilter -- SuggestionFilter

Change query filter

Name:

Filter type: ▾

Options:

Order:

[✖ Delete](#)

Figure 10: Django admin interface for customising an individual query filter

As Figures 6—8 show, non-programmers should be able to make changes to a service without needing to go near any code, allowing multiple configurations to be rapidly created and evaluated.

7 Summary

This report outlines the current map of the framework, showing the main trunk of development and the various branches that have emerged throughout the progress of the project; the current set of features that are included in the core framework release; a brief overview of the public documentation and its contents; the dependencies and installation requirements to get started with the framework; and the main developments that will be incorporated into the third and final release with the project lifecycle. The next deliverable will be D4.7 – Release of the Open Source Framework V3.0.

8 References

- [1] **Annex 1: Description of Work**, PuppyIR, 2008
- [2] **D1.2 – Agreed User Requirements and Scenarios"**, PuppyIR, 2009
- [3] **D1.3 – Agreed Technical Requirements**, PuppyIR, 2009
- [4] **D3.1 – Report of Data Pre-processing**, PuppyIR, 2009
- [5] **D4.1 – Specification Report**, PuppyIR, 2010
- [6] **D4.2 – Design Report**, PuppyIR, 2010
- [7] **D4.3 – Report on Implementation and Documentation**, PuppyIR, 2010
- [8] **D4.4 – Release of the Open Source Framework V1.0**, PuppyIR, 2011
- [9] **D4.5 – Report on Design and Specification Changes**, PuppyIR, 2011
- [10] **Django Project** (Web Application Framework): <http://www.djangoproject.com/>