D4.3 Report on Implementation and Documentation

File name	PuppyIR-D4.3-Report-on-Implementation-and-Documentation-v1.0
Author(s)	Richard Glassey (UGLW) Leif Azzopardi (UGLW) José Miguel Garrido (ATOS) Jeldrik Schmuch (ATOS) Paul Moore (ATOS)
Work package/task	WP4
Document status	Final
Version	1.0
Contractual delivery date	30 September 2010
Confidentiality	Public
Keywords	Implementation and Documentation
Abstract	This report describes the progress of the implementation and documentation efforts for the PuppyIR Framework.

Table of Contents

Executive Summary

This report details the progress of the implementation and documentation efforts for the PuppyIR Framework **[1]**. The implementation of the framework has been guided by the following 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 **[7]**
- D4.2 – Design Report **[8]**
- Report on Implementation of Prototypes and Demonstrators (WP7) **[Draft]**

This report provides a summary of the technology choices (options initially outlined in **D4.2;** Python, Django, jQuery, MT4J and OpenSearch) that the development team (spread across several work packages) have decided upon as the most appropriate set to meet the requirements of the project and scenarios.

In combination with the requirements (**D1.2** and **D1.3**), specification (**D4.1)** and design **D4.2**), the feasibility of implementation decisions has been investigated by creating a series of functional prototypes (FiFi and SeSu). They have helped to test and improve initial design decisions, and confirm that the implementation technologies are a good fit for the framework.

The state of the initial framework is also presented, reviewing the core modules (Service, Model, Query, Search, and Results) that have been implemented. The report concludes with a series of 'getting started' guides, documentation on how to build a basic service, and a tentative release plan for the first version of the framework (**D4.4 – Release of Open Source Framework – v1.0**).

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]**. A major component required to achieve this aim is the development of an open source framework that allows systems developers to make new information services, specifically for children, by reusing and extending a set of dedicated components. The purpose of this report is to provide an update of the implementation and documentation work conducted so far towards the PuppyIR Framework (henceforth referred to as the 'framework'). The goals of this report are to:

1) Brief the reader on the range of technologies that the framework builds upon
2) Describe two prototypes developed so far
3) Provide details on the evolution of the framework's software architecture
4) Provide documentation and tutorial material for using the framework
5) Outline the plan for reaching the initial release (v1.0)

1.1 Audience and Scope

This report serves three key purposes:

1. Provide a summary of implementation and documentation efforts undertaken by members of the project responsible for the software development (the developers) of the framework and its components.

2. Describe the initial parts of the framework that have been implemented to assist in integration efforts between separate work packages

3. Communicate to all project members and stakeholders how the project is progressing from the software design description towards the open source framework that will facilitate the construction of child-friendly information services.

The scope of the report is limited to describing the main technologies that have been chosen, rather than a complete description of everything that has been investigated so far (e.g. technologies deemed not useful in the implementation effort). **D4.2 – Design Report [8]** included a list of potential options for reference. Also, the tutorial and documentation material presumes a level of technical competence. Where supplementary material exists, it is referenced.

1.2 Implications from D4.2 – Design Report

In **D4.2 – Design Report**, a detailed design of the framework was presented, using a collection of standard software description views (**Composition, Logical, Information, Interface, and Interaction**). The composition and logical viewpoints described the initial class diagram of the components of the framework (**Admin**, **Client**, **Query**, **Results** and **User**). The Client, Query and Results components have been implemented and there are no major deviations from the original design. An extra **Search** component has been created by extracting parts of Query and Results components. There are no new classes, just a reorganisation of the component boundaries. As the development of the framework is early in its implementation phase, there is still opportunity to refine and improve the design. Other components, Admin and User, have not yet been implemented within the framework. Deviations from the design will be reported as part of the future deliverable, **D4.5 – Report on Specifications and Design Changes**.

1.3 Technology Glossary

The following abbreviations, technology and definitions are used in this report:

Term	Definition
WAF	Web Application Framework – a suite of software that supports the construction of web based applications that are normally accessed via a web browser.
Ajax	Asynchronous JavaScript and XML – a collection of technology that enables dynamic updating of web pages without needing to reload.
Django	A Python based WAF.
Multi-touch	A device that responds to multiple touch input from a user(s).
MT4J	A multi-touch framework for the Java programming language.
OSS	Open Search Standard – a common XML data format for describing a search service, search queries and search results.
JSON	JavaScript Object Notation – a compact and convenient data format often used for AJAX based web applications.
jQuery	An AJAX library for building dynamic web applications.
REST	Representational State Transfer – an approach to using existing HTTP methods to structure the API of a web service or application.

1.4 Structure

The remainder of the design report is divided into the following major sections:

- **Section 2 – Technologies**
- **Section 3 – Prototyping the Framework**
- **Section 4 – Implementing the Framework**
- **Section 5 – Getting Started Guides**
- **Section 6 – Building a Basic PuppyIR Service**
- **Section 7 – Release Plan for v1.0**
- **Section 8 – Summary**

2 Technologies

The purpose of this section is to provide an overview, justification and description of the technologies that have been chosen to build the framework from. As discussed in **D4.2**, the approach adopted was to use a web application framework (WAF) as the base to build the rest of the framework components around. The following sections will discuss these choices from the system, user interface, data format and logging perspectives. The section concludes with a discussion of some future options that may benefit the framework.

2.1 Overview of Technologies

Throughout D4.2, potential technologies were identified that would be useful in constructing the framework. Each design viewpoint section concluded with a survey of candidate technologies or standards that would be suitable for use. In the composition viewpoint (which looked at the decomposition of the framework at a high level) a range of server-side web application frameworks were suggested, each implemented in a different programming language. Technologies for supporting dynamic and engaging interfaces on the client-side, such as jQuery for desktop-based web applications and MT4J for multitouch device applications, were also identified. Finally less tangible technology decisions, such as standards based data-formats and design patterns were identified.

The following sections report the options that have been chosen for the implementation of the framework components and used for the development of initial prototypes.

2.2 System

System technologies are those that execute on the server-side of a PuppyIR service. The most elemental server-side choice is the programming language. The next section motivates the choice of the Python programming language.

2.2.1 Python

Python is a high-level, multi-paradigm, open source programming language that emphasises code readability and flexibility **[9]**. It is a popular scripting language that has been undergoing consistent development since 1991.

It is relatively simple to learn for experienced programmers; efficient in terms of its syntax; and easily lends itself to rapid prototyping and experimentation, due in part to its convenient interpretative mode and comprehensive library of modules.

As a large proportion of work in PuppyIR is research-driven and collaborative, Python is a natural choice. Its syntax (indentation indicates block structure) helps enforce a greater level of consistency in code appearance, compared to languages that use brackets for block structure. Given the distributed development team spread across Belgium, Scotland, The Netherlands and Spain, it is useful to choose a language that lends itself to readability and comprehension.

In recent years, with the emergence of many web application frameworks, Python has been a popular choice for implementation. By choosing Python for PuppyIR, the next decision of which web application framework to choose is made quite simple, as the community mostly recommends the Django framework. The next section looks at some more details on why this is a suitable choice for the PuppyIR framework.

2.2.2 Django

Django **[10]** is an open source web application framework, implemented in Python, that follows the model-view-controller pattern. It focuses on providing an easy to use framework that allows

web sites to be built quickly and with minimal effort. It comes with its own development web server, avoiding the need to deploy a production ready web server environment during development, making it easy to share with other developers and testers across multiple platforms. Various Django components are bundled to simplify the task of site administration, user authentication, and generating RSS/Atom feeds. Furthermore, there is an active Django community that produces high quality extensions to the core framework components (e.g. providing web service interfaces).

For PuppyIR, the web framework becomes the middleware platform on which various services are built. Django is an efficient choice as it provides many pre-built components that should speed up the development of the PuppyIR framework. This creates more development time to focus on the novel aspects of the project, such as the user interfaces and the various components for creating child-friendly information services, such as query processing and suggestion, as well as various document filters and recommenders that are specifically designed for children.

2.3 User Interface

The user interface technologies are those used on the client devices that children will use to access PuppyIR services. Project scenarios **[see D1.2]** have focused mostly on traditional desktop-based platforms and also on multitouch devices such as tables and tablets to access PuppyIR services. This decision requires the client component of the framework to support these platforms, whilst also being flexible enough to support other platforms (e.g. mobile phones).

2.3.1 jQuery

In recent years, Ajax has emerged, both as a collection of technologies and as a methodology that yields dynamic, responsive and engaging web application user interfaces. One particular library that has become dominant is jQuery **[11]**. jQuery provides several key features: handling client/server communication using the XML HTTP Request Object channel; providing numerous visual effects and transitions; and vastly simplifying the task of programming in JavaScript and processing the Document Object Model (DOM). Due to its plug-in-architecture and active developer community, a wide range of extensions are available to further enhance the development of dynamic and engaging user interfaces.

2.3.2 Multitouch for Java

There has been a substantial increase in the number of multitouch devices in recent times. No longer a specialised device, they are gradually pervading most environments where previously only traditional desktop machines existed. The simplified touch interface has undoubtedly increased the potential for user engagement with information services.

Alongside the rise of multitouch devices, there has been an increase in the number of frameworks for simplifying the development of multitouch-based applications. Multitouch for Java (MT4J **[12]**) is an open source framework that can be deployed on a wide range of platforms (tested for Windows 7, Vista, XP, Ubuntu, and Mac OS X). It supports the creation of new multitouch gestures on top of the existing common library, includes prebuilt user interface widgets and supports a wide range of visual media, including animating objects.

2.4 Data Formats

As the framework is expected to integrate with external information services, such as search engines that provide an application programmers interface (API), it must handle the common data formats that these results are encoded in. As stated early in the development of the project, OpenSearch **[13]** is a standard identified as being an ideal way for PuppyIR services to expose themselves to 3rd parties (and other PuppyIR services). It is mandatory therefore to support it as

an external format, use it as the main influence of the internal data model and support it as an export format.

2.4.1 OpenSearch

OpenSearch is a standardised method of exposing the search (and suggestion) functionality of a service **[14]**. It is an open standard developed by Amazon that allows clients to discover if and how a service can be searched, and it specifies a standard format for result feeds to be encoded in (based upon RSS/Atom feeds). As an example, the Firefox web browser search box is implemented as an OpenSearch client. If a web page is visited that supports OpenSearch, then Firefox can be add the search engine to its search box.

Search Description in OpenSearch:

```
<?xml version="1.0" encoding="UTF-8"?>
 <OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">
   <ShortName>Web Search</ShortName>
   <Description>Use Example.com to search the Web.</Description>
   <Tags>example web</Tags>
   <Contact>admin@example.com</Contact>
   <Url type="application/rss+xml"
        template="http://example.com/?
        q={searchTerms}&amp;pw={startPage?}&amp;format=rss"/>
 </OpenSearchDescription>
```

Search Response in OpenSearch:

```
<?xml version="1.0" encoding="UTF-8"?>
 <rss version="2.0"
      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
      xmlns:atom="http://www.w3.org/2005/Atom">
   <channel>
     <title>Example.com Search: New York history</title>
     <link>http://example.com/New+York+history</link>
     <description>Search results for "New York history" at Example.com
     </description>
     <opensearch:totalResults>4230000</opensearch:totalResults>
     <opensearch:startIndex>21</opensearch:startIndex>
     <opensearch:itemsPerPage>10</opensearch:itemsPerPage>
     <atom:link rel="search"
                type="application/opensearchdescription+xml"
                href="http://example.com/opensearchdescription.xml"/>
     <opensearch:Query role="request"
                       searchTerms="New York History"
                       startPage="1" />
     <item>
       <title>New York History</title>
       <link>http://www.columbia.edu/cu/lweb/eguids/amerihist/nyc.html</link>
       <description>
         ... Harlem.NYC - A virtual tour and information on
        businesses ...  with historic photos of Columbia's own New York
        neighborhood ... Internet Resources for the City's History. ...
       </description>
     </item>
   </channel>
 </rss>
```

2.4.2 JavaScript Object Notation

Besides OpenSearch, many search engines return search results in non-standard, proprietary XML and JSON formats. JSON, JavaScript Object Notation **[15]**, is a very compact representation, primarily used as convenient shorthand for instantiating JavaScript objects. Recently, it has gained a lot of popularity as most consumers of web APIs are services running within a web browser, and have immediate access to a JavaScript interpreter.

JSON model of a person, address and numbers.

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address":
     {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber":
     [
         {
           "type": "home",
           "number": "212 555-1234"
         },
         {
           "type": "fax",
           "number": "646 555-4567"
         }
     ]
 }
```

2.4.3 Internal Data Model

Once a response from an external search service has been received, it is transformed from its OpenSearch, XML, or JSON representation into an internal Python object model to allow various framework components to act upon it. The design for this model reflects the OpenSearch model, and such, consists of a Query, Response and Description object. Section 4.3 describes this model in detail with examples.

2.5 Logging

Although not of immediate concern for development, initial efforts have been made to identify logging toolkits that may be integrated into the framework to generate data for research purposes. It should be noted that the PuppyIR project supplies a reusable ethics guide for service developers to use, see **D1.5 – Ethics Manual**.

2.5.1 UsaProxy

UsaProxy **[16]** is a research project that is investigating the usability of web interfaces. As part of this work a software tool was created that logs all of the JavaScript events in a web browser and transmits them to a remote server. The crucial advantage of UsaProxy is its non-intrusive nature. It operates by intercepting an http request (at a proxy site) and decorating the response with its event listener. This requires no modification of the web application and no modification of the client's browser.

2.5.2 Server Logging

Besides client-side interaction logging, it is likely that various aspects of the server will be logged for analysis. Python has many logging libraries including Log4Py **[17]** and Logbook **[18]**.

2.5.3 Multitouch

This is an unexplored area at present and future work will report the available methods for multitouch logging.

2.6 Future Options

The one area that is most likely to be further explored is the range of technologies that are available for building user interfaces. At present only the desktop and multitouch platforms are being considered here. Future work could investigate what PuppyIR Services might be like on mobile and other immersive environments (e.g. smart class rooms).

3 Prototyping the Framework

The purpose of this section is to report the development of two prototype services, and describe how the experiences have helped influence the implementation of the framework. In software engineering, there is a rule of thumb for a framework, stating that it should be possible to construct three distinct applications from its components. This acts as a good guide of the framework's flexibility and re-usability to software developers. In light of this, it was decided to build two distinct prototypes before beginning development of the core framework components. Then, once the framework was more complete, a further two demonstrators will be built. The following sections describe the initial prototypes.

To kick-start the development of the framework and to test the viability of the design choices, two prototypes have been implemented. The first prototype (FiFi) is a novel news filtering application for children. The second prototype (SeSu) is a query suggestion service built to integrate components developed elsewhere in the project work packages. Both these prototypes provide practical experience of the technology choices detailed in the previous section.

3.1 Motivation

Frameworks tend to be especially complicated to design and develop. They must be written at an appropriate level of abstraction that is flexible enough to be used for multiple applications, yet be domain-specific enough to be useful in building specialised applications. More often than not, they attempt to take away the drudgery of repetitive tasks in software development. For example, web application frameworks all attempt to reduce the effort involved in making database connections within the application logic.

Some frameworks are designed from the outset, others emerge from projects where the same applications tend to be required, and it is efficient to identify the commonality amongst these applications and move it into a framework. For PuppyIR, the framework has been designed first. However, this does not exclude us from testing our ideas with prototypes as the framework is developed. The duel approach has the benefit that we discover our mistakes earlier in the development effort and avoid major redesign at a later stage.

3.2 FiFi – Find and Filtering News for Children

The first prototype developed was FiFi, a novel news filtering service that was designed specifically for children. A cursory search of news services for children revealed that this was a poorly supported area for children. Existing solutions were either too basic (e.g. CBBC Newsround), or not appropriate for children (Google News).

3.2.1 Purpose

Besides filling a gap in children's online news services, the following objectives were set:

- Investigate Django as middleware
- Link Django to the Lemur search engine
- Build a collection of children's news from RSS feeds
- Create an engaging user interface experience with jQuery

3.2.2 Design

The high level design of FiFi is illustrated in Fig. 1.

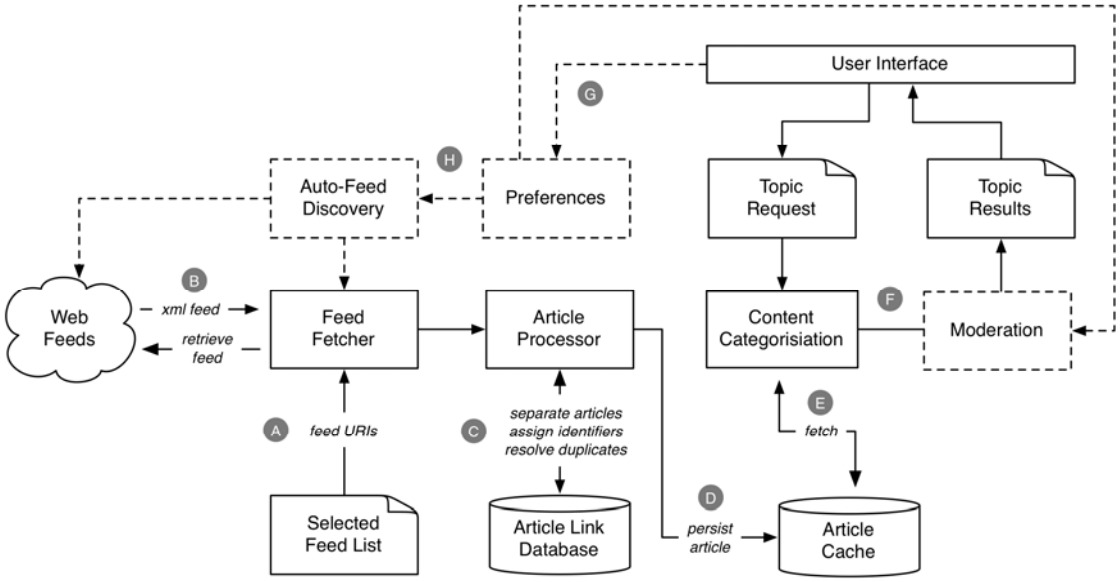

Figure 1 - FiFi System Design and Flow

A list of news feeds is specified as a list of URLs (A). The FeedFetcher component (implemented using the Rome Framework for RSS and Atom feed processing **[19]**) periodically retrieves the contents of the feeds and splits each feed into a collection of entries (B). Each entry is checked against a database of previously retrieved entries to determine whether an incoming entry is unique (C). If an incoming entry is unique, a new row is created in the database, including the entry URL and timestamp. The row ID is then used to represent the entry position in the order of entries retrieved. The content of the entry is then written to disk (D), making it available for the content categorisation component.

The content categorisation method used in FiFi is based on an information retrieval approach using the Lemur search engine to retrieve recent articles that match a topic request (E). The articles are converted to JSON and sent to the user interface (F). The user interface displays topic terms to represent areas of interest. These terms are used to search the indexed collection of entries that have been collected on disk so far. Crucially, as this is a news service and time-order is important, the retrieved entries for a term are ordered not by relevance, but by date. However, the relevance scores are preserved as metadata for the entry so that the user interface can highlight older but more relevant entries.

The user interface design of FiFi is illustrated in Fig. 2. It is implemented using jQuery and is presented via a web browser.

The set of default topics are shown above the set of discovered topics (1a). In this example, the Science topic has been selected (1b) and the list of document titles filtered for this topic is presented to the child. The amount of space used in presenting a title indicates the relevance of a document, as shown by comparing a somewhat relevant document (2a) and a strongly relevant document (2b). Children are also able to manually define a new topic (3), if needed, and personalise the interface (4), such as the customising the title, colour scheme and style.

When a child clicks on an item in the list, for example (2b), the remainder of the document is presented (not shown). This interaction is used to update the system's knowledge about the child's interests. The history of interaction actions is subsequently used to learn the set of discovered topics (1a), to provide a personalised experience.

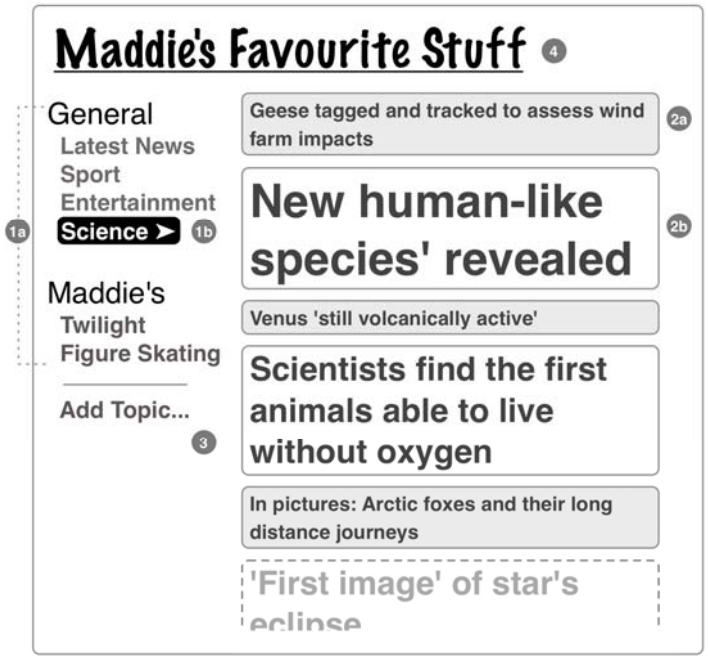

Figure 2 - FiFi User Interface Design Mockup

3.2.3 Outcomes

The major outcome of this work was that Django was more than capable of acting as the middleware glue for the many different aspects of this work. Integration with Lemur was made straightforward by the discovery of the Pymur project **[20]**, which provides a Python-based wrapper for the Lemur API. Integration with jQuery was also straightforward and presented no concerns as a design choice. jQuery itself enabled a very accurate rendition of the user interface design to be realised, that was both attractive and simple to use, without being too basic. Finally, the FeedFetcher has now been indexing a collection of children's and adult's news services for ten months and will be made available to be used within future parts of the project that require such a collection.

3.3 SeSu – Search and Suggest for Children

The second prototype developed is SeSu, an experimental service to provide appropriate search suggestion for children. As adults tend to use short search queries, search suggestions are an effective means of improving some queries. However it is unclear if children would benefit from search suggestions, and what types of suggestion would best assist them. This service aims to provide useful suggestions to children by using term expansion with their queries as well as presenting suggestions via the interface.

3.3.1 Purpose

Building on the success of FiFi, much of the initial components were reused and repurposed for SeSu. The main objectives of this prototype were:

- build query/result processing pipes
- integrate with external search engines
- develop query processing components
- develop result processing components

3.3.2 Design

The high level design of SeSu is illustrated in Fig. 3.

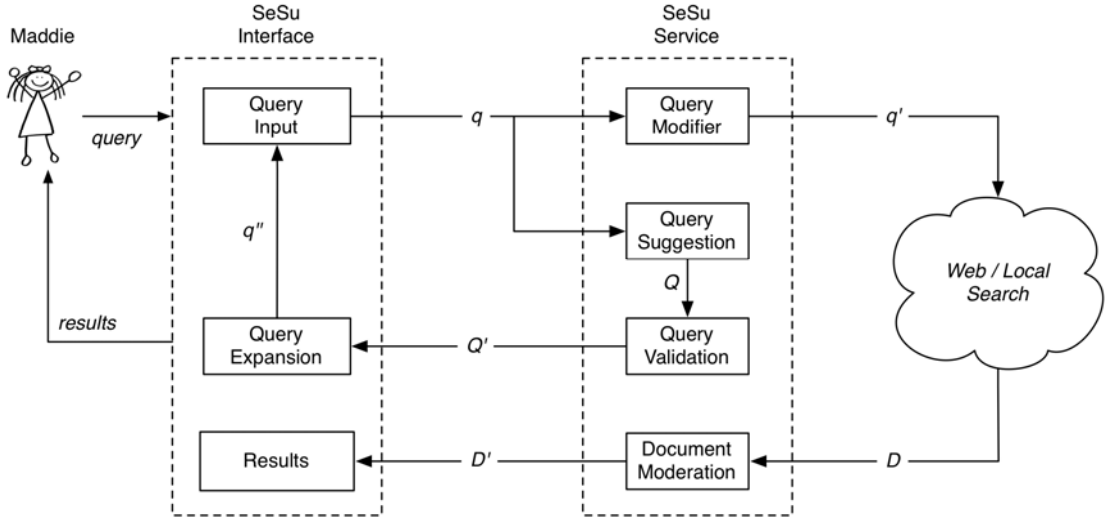

Figure 3 - SeSu System Design and Flow

The user, Maddie, issues a query using the SeSu interface. This query is received by the SeSu service. The service is configured to pass the query through QueryModifier and QuerySuggestion components. The QueryModifier component simply extends the query terms. For example, a useful expansion is simply "for kids". This modified query is then dispatched to a specified search engine (e.g. Google and Yahoo!), which can be configured to use a 'safe mode' if supported (e.g. Yahoo! supports the flags '-hate-porn', whilst Google has a 'Safe Search' mode).

The QuerySuggestion component takes the query as input and returns a list of queries that are semantically related terms that aim to improve retrieval of subtopics. The query suggestions are displayed underneath the original query box. In some cases, these may be nested and require pop-ups to appear to reveal the extra choices to the user.

3.3.3 Outcomes

Having already built FiFi, it was much easier to create a completely distinct service, but reuse some of the components already developed. Some components were not required, such as the FeedFetcher (no RSS processing required) and Lemur integration (search was handled remotely by a search engine). The main progress in the development of the framework was the establishing of pluggable pipelines for introducing query and result filters into the service. This represents one of the key contributions the framework will make once a collection of filters have been developed, allowing a developer to quickly experiment with different configurations without having to completely re-engineer their system. The same applies with search engines, as it is now a relatively trivial task to incorporate a new online search service via a common interface.

3.4 Summary

Both prototypes have each been instrumental in motivating and guiding the development of the initial core components of the framework. Design choices and technology options have been tested, and the results have been fed into the development process. The next section will review the core components of the framework that are now available on the Sourceforge repository.

4 Implementing the Framework

The development of the prototypes has provided the opportunity to implement an initial version of the framework. The following sections look at the Service, Model, Query, Search and Results components. As mentioned in the Introduction, the Search component has been introduced since the initial design **D4.2** by removing some classes from the Query and Results components and grouping them together more logically.

The framework is divided into the following major modules: Service, Model, Query, Search, and Results. The following sections are generated from the documentation embedded into the existing codebase.

4.1 puppy.service

This module is for creating new search services, creating query filter pipelines, performing query suggestion, specifying which search engines to use, and creating result filter pipelines. Also, site-specific configurations can be made for network proxies and individual API keys for search engines that require them. For example:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from puppy.service import Service
from puppy.search import SearchEngine
from puppy.query.filter import TermExpansionFilter
from puppy.query.filter import SuggestionFilter
from puppy.query.filter import SuitabilityFilter
from puppy.model import Query

service = Service()

service.add_query_filter(TermExpansionFilter("--terms=for+kids"))
service.add_query_filter(TermExpansionFilter("--terms=colouring+book"))
service.add_search_suggestion(SuggestionFilter())
service.add_result_filter(SuitabilityFilter())
service.add_search_engine(SearchEngine('Pathfinder'))

query = Query('elmo')
service.search(query)
```

---

class puppy.service.Service.Service [source]

A PuppyIR Search Service.

Establishes the configuration of the QueryFilter pipeline, QuerySuggestion pipeline, Search Engine, and the ResultFilter pipeline.

search(query)[source]

Run a search on the SearchService.

Parameters:

query (Query): user query

Returns:

results (puppy.model.OpenSearch.Results): search results

4.2 puppy.model

This module is for the data model used by search services. It is based on a subset of the OpenSearch Standard (e.g. Description, Query and Response), and can parse OpenSearch Responses and write new OpenSearch Responses from results gathered from a non-compliant search engine.

There is planned support for handling OpenSearch Queries and creating OpenSearch Description documents (n.b. Query is already used internally as the representation of a search query by components).

4.2.1 OpenSearch.Query

Models a query within the search service. At present this is a fairly trivial class that models some useful query attributes (e.g. search_terms, start_index, language, service, and a map of suggestions).

class puppy.model.OpenSearch.Query(search_terms)[source]

OpenSearch Query.

Models an OpenSearch Query element.

See:
http://www.opensearch.org/Specifications/OpenSearch/1.1#OpenSearch_Query_element

static parse_xml(oss_xml)[source]

Parse OpenSearch Query XML.

Parameters:

oss_xml (str): OpenSearch Query XML

Returns:

puppy.model.OpenSearch.Query
TODO code Query.parse_xml()

write_xml()[source]

Creates XML for OpenSearch Query.

Returns:

query_xml (str): OpenSearch Query as XML
TODO code Query.write_xml()

4.2.2 OpenSearch.Response

Models a Response from a search engine to a Query. Can be created from the parse_xml() factory method or built manually by populating its attributes (e.g. title, description, total_results, items_per_page, and a list of Items).

class puppy.model.OpenSearch.Response[source]

OpenSearch Response.

Models an OpenSearch Response element.

See: http://www.opensearch.org/Specifications/OpenSearch/1.1#OpenSearch_response_elements

static parse_xml(oss_xml)[source]

Parse OpenSearch Response XML.

Parameters:

oss_xml (str): OpenSearch Response XML

Returns:

response (puppy.model.OpenSearch.Response)

write_rss()[source]

Creates an RSS feed from a OpenSearch Response.

Returns:

response_xml (str): OpenSearch Response as XML

4.3 puppy.query

A search service can be decorated by one or more query filters that process a query before it is dispatched to one or more search services. Lists of query suggestions can also be generated which are returned to the user.

4.3.1 Abstract QueryFilter

All filters extend the abstract QueryFilter class. The main run() method takes a Query object as its argument and returns the same object. Subclasses are expected to provide the query processing logic within this method. Filters can also be passed a string of parameters (much like a unix command-line application).

class puppy.query.QueryFilter.QueryFilter(options='')[source]

Interface for query filters.

parse_options(options)[source]

Parse options for filter.

Filters may expose an arbitrary number of options that can customise their behaviour. The options string can contain any number of long form unix command line options (e.g. - -time=10) that are separated by a space.

Parameters:

options (str): a string of unix command line options to be parsed

Returns:

parsed_options (dict): a map of available options for this filter

Raises:

Exception: caused by malformed option

run(query)[source]

Filters the query.

Parameters:

query (puppy.model.OpenSearch.Query): query to be filtered

Returns:

query (puppy.model.OpenSearch.Query): filtered query

4.3.2 Implemented QueryFilters

Part of the contribution of the PuppyIR project is novel components to build child-friendly information services from. The implemented query filters below are the initial contribution in the area of query support that is specifically aimed at children.

- **SuggestionFilter** - Creates a set of suggestions based upon the query search terms.
- **TermExpansionFilter** - Expands original query terms with extra terms.

4.4 puppy.search

Another area of PuppyIR's flexibility is in which search service it uses. In the initial version of the framework, four search services are supported: Bing, Yahoo!, Google, and Pathfinder (a search engine from our partner Emma Hospital). As well as these services, there is a general purpose OpenSearch discovery and search service for any OpenSearch compliant service.

4.4.1 Search Engine Factory

This class is responsible for taking an argument that specifies which search engine to create. If an implementation of a search service exists, it will be used; otherwise an attempt will be made to create an OpenSearch engine.

class puppy.search.SearchEngine.SearchEngine(engine)[source]

Generic search engine interface.

SearchEngine configures urllib2 with the PROXYHOST setting, attempts to load the specified search engine, and if unavailable attempts to discover an OpenSearch compliant search engine.

configure_opener()[source]

Configure urllib2 opener with PROXYHOST.

load_engine()[source]

Load specified search engine, or attempt to discover an OpenSearch compliant service from a URL.

Returns:

puppy.search.engine module (e.g. Yahoo.py)

search(query)[source]

Perform a search using the specified engine, or OpenSearch compliant service.

Parameters:

query (puppy.model.OpenSearch.Query): the query object

Returns:

puppy.model.OpenSearch.Response: the results of the search

Raises:

Exception: a search engine was not specified

4.4.2 Implemented Search Engines

The following search engines are currently supported:

- Bing
- Google
- OpenSearch
- Pathfinder
- Yahoo!

4.5 puppy.result

As a complement to the query filters, result responses can be processed by a number of components extending from the abstract ResultFilter class.

4.5.1 Abstract ResultFilter

class puppy.result.ResultFilter.ResultFilter(options)[source]

Interface for search result filters.

parse_options(options)[source]

Parse options for filter.

Filters may expose an arbitrary number of options that can customise their behaviour. The options string can contain any number of long form unix command line options (e.g. - -time=10) that are separated by a space.

Parameters:

options (str): a string of unix command line options to be parsed

Returns:

parsed_options (dict): a map of available options for this filter

Raises:

Exception: caused by malformed option

TODO substitute spaces with plus in parsed_options(options)

run(results)[source]

Filters the set of results.

Parameters:

results (puppy.model.OpenSearch.Response): results to be filtered

Returns:

results (puppy.model.OpenSearch.Response): filtered results

4.5.2 Implemented ResultFilters

- **SuitabilityFilter** - Filters search results based on the results' suitability for children.

4.6 Summary

The documentation of this section has been a view into the development of the framework and how it is being structured around the core components for specifying a search service, the query processing, the search engine, the data model and the result processing.

5 Getting Started Guide

This section provides a basic introduction to the core technologies used for the framework and provides references to more detailed external documentation.

5.1 Overview

PuppyIR uses the following set of programs, frameworks and libraries all of which are published under Creative Commons Licenses:

- Python
- Django
- jQuery
- MT4J Multitouch for Java
- Gstreamer
- Moszwing

5.2 Technology Setup

Given the range of differing technologies selected, their individual configuration and integration can be problematic. The following section provides some insight gained into dealing with the tricky issue of software setup.

5.2.1 PuppyIR Multitouch Software and Applications

The following instructions are based on the assumption that Java 6 Development Kit and a working version of the Eclipse IDE are installed (other IDEs can be used, but the instructions below would vary somewhat).

MT4J is the most important element for the multitouch application. MT4J is a framework for creating visually rich applications, with focus in multitouch support. MT4J is open source and programmed using Java (J2SE). One of its key advantages is that MT4J is cross-platform; currently it is tested under Windows 7, XP, Linux and OS X.

When developing with the MT4J framework it is advisable to deploy and test software on a multitouch device. However, it is possible to run using one or more point devices (e.g. a mouse) to simulate multitouch on a traditional computing platform (e.g. simple desktop setup). The "multimouse"- functionality needs to be activated in Settings (see: `mt4j/settings.txt`) by changing "MultiMiceEnabled" to "true". Furthermore, display resolution and full screen mode can be configured here too.

To use MT4Js video player class, the GStreamer library is needed. GStreamer is a multimedia library written in C. GStreamer has been ported to the several operative systems, like MS Windows, Linux, Solaris, Mac OS X or Symbian.

In the most common case (developing in MS Windows environment) the following software has to be installed:

- Gstreamer (http://forja.rediris.es/frs/download.php/1207/GStreamer-WinBuild-0.10.3.exe)
- Microsoft Visual C++ 2008 SP1 Redistributable Package (x86) (http://www.microsoft.com/downloads/details.aspx?FamilyID=a5c84275-3b97-4ab7-a40d-3802b2af5fc2&DisplayLang=en)
- GTK (http://gtk-win.sourceforge.net/home/index.php/en/Downloads)

On some systems, Gstreamer may not run correctly as expected. In these cases, deleting `libgstspeex.dll` from `gstreamer/lib` could be a possible solution to solve the problem.

The PuppyIR multitouch desktop application provides classes for using three different Browser types:

- Lobo Browser
- JDIC Browser
- Mozswing Browser

Lobo Browser is a lightweight cross platform Web browser entirely written in Java, with support for HTML 4, JavaScript and CSS2. Being independent from system installed Web browsers makes it highly portable, but speed issues and a lack of Adobe Flash support limit its usability.

JDIC Browser is part of the JDesktop Integration Components that attempts to provide access to functionality and facilities of the native operation system. It provides a fast and fully functional Web browser, making use of either Internet Explorer or Mozilla (including all installed plug-ins) that are already installed on the system. However this does restrain the platform independence aspect.

Mozswing Browser integrates the Mozilla rendering framework XUL with the Java Swing GUI framework. It supports all installed plug-ins, but also relies on an installed version of Mozilla Firefox being in place.

All libraries and source files of Lobo Browser and JDIC Browser are already integrated into the framework. For use of the Firefox based Mozswing Web Browser the latest built has to be downloaded from the Mozswing project site (http://mozswing.mozdev.org) its containing folders being copied to `mt4jLibs/mozswing`.

5.2.2 Python & Django

Python is a high level multiple platform programming language supporting multiple programming paradigms. It ships with a large and comprehensive standard Library and is published under the GPL like “Python Software Foundation License. Version 3.1.2 is the latest Release Candidate, however because of backwards compatibility some frameworks (including Django) only support Python version 2.7.

For PuppyIR Python 2.6 or 2.7 (www.python.org/download/) and Beautiful Soup (http://www.crummy.com/software/BeautifulSoup/), a free HTML/XML parser for python have to be downloaded and installed.

Django is a high-level Python Web framework that provides abstraction of common Web-development patterns. It provides a powerful set of tools for creating stable, secure and easy to maintain websites supporting all four major database engines.

The latest release could be found at http://www.djangoproject.com/download/.

To install Django onto a Unix System, the following steps are required:

- tar xzvf Django-1.0.2-final.tar.gz
- cd Django-*
- sudo python setup.py install

On Windows platforms, archive and extraction software (unzipping), such as the popular 7-Zip utility (http://www.djangoproject.com/r/7zip/), can be used for unzipping .tar.gz files required for installation. Once unzipped, Django can be installed by typing “python setup.py install” from within the directory that starts with “Django-:” using a DOS command prompt.

5.2.3 jQuery

jQuery is a cross-browser JavaScript library published both under the MIT License and the GPL. It can be downloaded from http://www.jquery.com and, being just a single file containing all of its common DOM and Ajax functions, it does not have to be installed but just be copied into the working directory of the web-based interface.

Django and jQuery comprise a different role in PuppyIR prototypes. Django forms the backbone of web applications, it manages all the dataflow from the PuppyIR framework to the user interface and return, The prototypes are made using the Model View Controller paradigm and jQuery enhanced the View part using some Ajax techniques, So the Django template files are written using the jQuery library.

5.3 Summary

Additional information about system requirements, tutorials and HowTo's regarding the above described programs, frameworks and libraries can be found online.

MT4j:

- http://www.mt4j.org
- http://nuigroup.com/forums/

Python:

- http://www.python.org/
- http://diveintopython.org/

Django:

- http://www.djangoproject.com/
- http://groups.google.com/group/django-users

jQuery:

- http://docs.jquery.com/Downloading_jQuery
- http://www.impressivewebs.com/jquery-tutorial-for-beginners/

6 Building a Basic PuppyIR Service

This section will describe how to develop a simple PuppyIR service using the existing components of the framework. The service will initially be developed as a simple front-end to the Yahoo! BOSS search service. Subsequent refinements will introduce a query expansion component, a query suggestion component, and finally a results filtering component. The section concludes with a summary of building PuppyIR services.

6.1 Introduction

In this section we describe an example of public search service. Then we describe briefly how the current modules of the frame are implemented and how we can add new modules to the existing ones.

6.2 Using a Search Service: Yahoo! BOSS API

Making an innovative customized search service from scratch could be a very challenging task. Fortunately, some companies offer us their search infrastructure, so we can obtain the basic set of results from inside our program or webpage, and then customize the results in order to fit different needs.

One of the more useful public search services is Yahoo! BOSS **[21]**. Yahoo! BOSS (Build your Own Search Service) offers Yahoo! Search Service to developers. Using BOSS, we use the powerful indexing and ranking algorithms from a large search engine company for free. BOSS is interesting because there are almost no limits, Yahoo! doesn't limit the number of searches per day, and so a competitive service can be constructed using BOSS.

The first step for a BOSS developer is to obtain a general Yahoo! ID (if needed), and then, an App Key. We have to go to the BOSS page inside Yahoo! Developer Network **[22]** and then press "Get App Key". Then, we have to choose "New Web/Client App".

Figure 4: Yahoo! BOSS homepage

The last step is a registration form. It is important to mark "Browser based authentication".

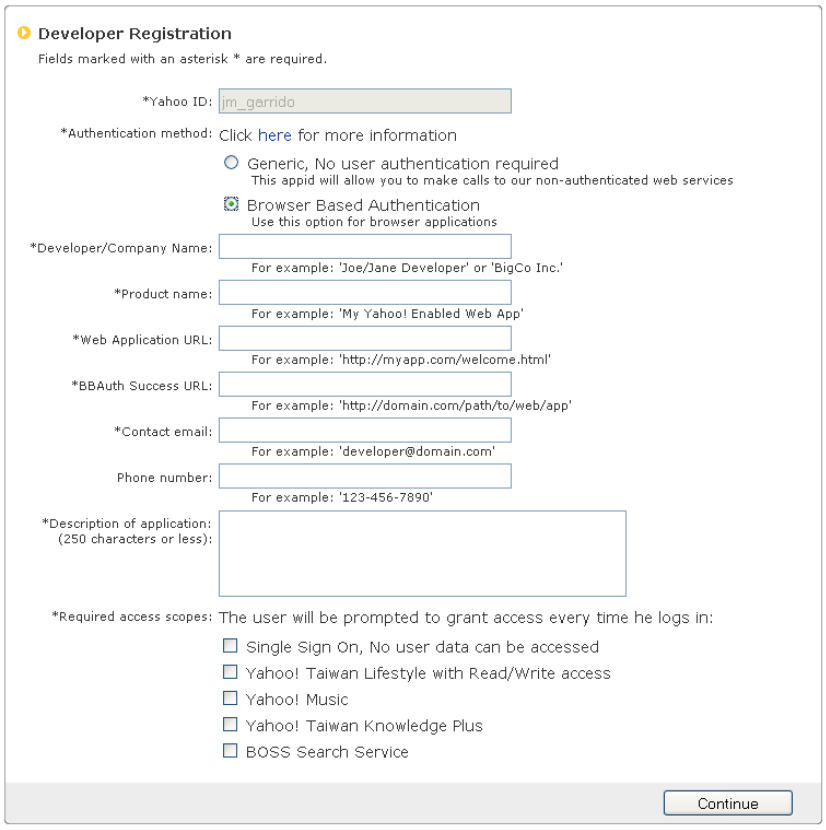

Figure 5: Yahoo! BOSS registration form

After completing these steps, Yahoo! will send you an e-mail with an ID. This ID is necessary for calling the BOSS API; you have to include it in every call.

Yahoo! BOSS offers us some different services:

- Web Search
- BOSS Site Explorer
- Image Search
- News Search
- Spelling Suggestions

For a search service, obviously the most interesting services are the Web, Image and News Search services.

It's quite easy to use the services utilizing different programming languages. The services are exported as SOAP-less web services. For demonstration purposes, it is even possible to call services from an Internet browser like Firefox.

The basic service is the Web Search. We can try an example of Web Search. For instance, we can search about "Elmo" just introducing this URL in a web browser.

http://boss.yahooapis.com/ysearch/web/v1/elmo?appid=ijsmWI.....WVFGQWxnKo-&format=xml&count=2

(it is mandatory to use a real appid for Yahoo! BOSS)

The BOSS service will return a response in XML format:

```
<ysearchresponse responsecode="200">
      <nextpage>
      /ysearch/web/v1/elmo?format=xml&count=2&appid=ijsmWI…FGQWxnKo-
      &start=2
      </nextpage>

      <resultset_web count="2" start="0" totalhits="3239367"
deephits="21500000">

      <result>
      <abstract>
            Welcome to Sesame Street! Play educational games and watch
            videos with <b>Elmo</b>, Abby, Cookie Monster and all your
            favorite Sesame Street friends. Engage your little
            <b>...</b>
      </abstract>
      <clickurl>
      http://lrd.yahooapis.com/_ylc=D...lQ-
      /SIG=111pu5s47/**http%3A//www.sesamestreet.org/
      </clickurl>
      <date>2010/09/18</date>
      <dispurl>www.<b>sesamestreet.org</b></dispurl>
      <size>27960</size>
      <title>Home - Sesame Street</title>
      <url>http://www.sesamestreet.org/</url>
      </result>

      <result>
      <abstract>
      .
      .
      .
      </result>

      </resultset_web>

</ysearchresponse>
```

The response is a set of individual “result” items. The more important elements of each “result” are:

- The title of the web page
- An abstract
- The target’s actual URL
- A “clickable” URL that we have to show to the user

It’s possible to choose between two different formats for the search results, JSON or a custom XML. Unfortunately, the XML format is not Open Search compliant, so a custom search module is necessary (see next section).

BOSS accepts many parameters for the search, so the service is very flexible. The more useful parameters are:

`Start`	Initial position for the search
`Count`	Total number of results to return. By default, it is 10.
`Lang`	Language to query: By default, English
`Region`	What regional (country) BOSS search service to query.
`Format`	Either JSON or XML.
`Sites`	Restrict search results to a set of sites
`Filter`	Filter out porn or hate sites. (hate filter only available for English)
`Type`	Specifies document formats, as pdf, text or msword

BOSS allows common search parameters to be used for constructing the search queries. It is possible to enclose the query between question marks for searching exact words or phrases, or to use the minus operator to exclude contents.

The Image Search service is similar to the Web Search service, but additional parameters are allowed. In the case of images, there is an Offensive Content Reduction Filter available.

The News Search service only searches in news-related sites. The age category of the results is relevant in this case, for instance, it is possible to order the results by age.

A complete reference can be found in **[22]**

6.3 Constructing a baseline no-op service

With this basic knowledge of a public search service as Yahoo! BOSS, we can start the tutorial about how the PuppyIR framework is built, and how a developer can add new services to the current ones. For this purpose, we will take the Yahoo! BOSS API wrapper as an example of implementation of a custom search module.

For Puppy developer, the first step is downloading the code from SourceForge Subversion repository. The current URL of the project in SourceForge is http://sourceforge.net/projects/puppyir/. There are some instructions in this page (made by SourceForge) about how to use Subversion. In the general case, it is sufficient to know that using the subversion standard tool; the command line could be like this:

```
svn co https://puppyir.svn.sourceforge.net/svnroot/puppyir puppyir
```

The use with other popular tools like Tortoise or the Eclipse plug-in must be straightforward.

The SVN repository is publicly available for downloading, so it is not necessary to be authorized by the project team or to provide a password of any kind.

The repository stores code for the framework and some prototypes. The framework source code is contained the directory `Framework`. As always in an open source project, the source code itself it is a valuable part of the documentation. If you are planning to add a new service to the framework, it is always advisable using the existing services as an example.

As it was stated in the Section 4, the framework is divided in several major modules:

- Model
- Service
- Query
- Search
- Results

The data types are defined in the Model module, in the file `OpenSearch.py`. The more important data types are Query and Response.

Query defines an object for the queries. The main data fields are:

- search_terms
- counts
- start_index
- start_page
- language

As you can see, there is a correspondence between these attributes of the object and the information given by Yahoo! BOSS or other commonly used search services.

The Response corresponds to the data available in the response of a public server. The most important element is a list (a python list) of Items objects. Each Item has:

- Title
- Description
- Link

These objects are prepared for the most common search engines, but it is possible to add new data items for new requirements, for instance, for the dimensions of an image.

In order to add a new search service to the framework, first it is necessary to check if the search service is OpenSearch compliant. Currently, there is an OpenSearch engine included in the framework, so in many cases, it not is necessary to program a new search engine.

If the new search engine is not OpenSearch compliant, we need to add a new file to the Search module. The new python file must be in the “engine” subdirectory of the Search module. The new file will be automatically loaded and executed, so the name of the file it is important. For instance, if we add a sentence like this to our program:

```
service.add_search_engine(SearchEngine('foo'))
```

then the file has to be called foo.py or the framework won't find it.

This file has to implement a mandatory function called “query” This function has to take as input a Query (from Model module) object and return a Response (also defined in the Model module). Usually, this function constructs a URL for the specific public search service from the data in the Query object. After calling the external web service, it is necessary to translate from the output format to the Response object.

For instance, for the Yahoo! BOSS engine (file `yahoo.py`) the translation to a suitable URL from the Query object is straightforward:

```
url = "http://boss.yahooapis.com/ysearch/
                  web/v1/{0}?appid={1}&format=json&
```

```
                style=raw&filter=-porn-hate&start={2}"
                .format(
                urllib2.quote(query.search_terms),
                YAHOO_API_KEY,
                str(pos))
```

There is another function in `yahoo.py` to make the translation from BOSS JSON format to the Response object. Using the library “simplejson” from Django, it must be straightforward.

The JSON answer from BOSS has the same information that the XML described in the previous section. The JSON is barely human-readable, but it is important to realize where the labels for the information fields are.

This is an example of a (summarized) JSON output from BOSS:

```
{"ysearchresponse":
      {"responsecode":"200",
      "nextpage":"\/ysearch\/web\/v1\/elmo?format=json&count=10&appid=i
j     WxnKo-&start=10",
      "totalhits":"3150653",
      "deephits":"20900000",
      "count":"10",
      "start":"0",
      "resultset_web":[
      {
            "abstract":"Welcome to Sesame Street! Play educational
            games and watch videos with <b>Elmo<\/b><b>...<\/b>",
            "clickurl":"http:\/\/lrd.yahooapis.com\/_ylc=X3oDMTU4ZZaw--
            \/SIG=111pu5s47\/**http%3A\/\/www.sesamestreet.org\/",
            "date":"2010\/09\/14",
            "dispurl":"www.<b>sesamestreet.org<\/b>",
            "size":"27960",
            "title":"Home - Sesame Street",
            "url":"http:\/\/www.sesamestreet.org\/"
      },
      {
            "abstract":"<b>Elmo<\/b> is a Muppet on the children's
            television show Sesame St<b>...<\/b>",
            "clickurl":"http:\/\/lrd.yahooapis.com\/_ylc=X3MUZaw--
            \/SIG=116i3v8l\/**http%3A\/\/en.wikipedia.org\/wiki\/Elmo",
            "date":"2010\/09\/11",
            .
            .
            .
      }
      .
      .
```

In the source code (`yahoo.py`), the first step is importing the library

```
from django.utils import simplejson
```

Using only one line, this library translates from a JSON-coded string to a python dictionary:

```
results = simplejson.load(response)
```

After doing this, `results` is an object indexed by the field names, so the translation to a `Response` object is straightforward.

```
response.total_results = results['totalhits']
response.start_index = results['start']
response.items_per_page = results['count']
for result in results['resultset_web']:
      item = Item()
      item.title = result['title']
      item.link = result['url']
      item.description = result['abstract']
      response.items.append(item)
```

There are more examples in the source code using different search engines: Google, Bing or general OpenSearch. The structure is similar in all cases.

6.4 Constructing a query expansion service

In the last section, we described how to develop a wrapper for a new public search engine. Another possibility of expanding the Puppy Framework is creating a new service. Building a new service, it is possible to expand the functionalities of the framework in innovative ways. The first topic about services is how a query expansion service is built, and how is possible to add a new one.

The query expansion service operates like a filter. It takes a Query object as input and returns another expanded Query object.

A new query expansion filter must derive from `QueryFilter (please see QueryFilter.py)`

```
class NewFilter(QueryFilter):
```

A query expansion filter must implement a constructor and a function called “run”.

The constructor takes some options as parameters. In the general, it is only necessary to call the “super” function in order to call the base class constructor, and then to change the `info` about the new service

```
def __init__(self, options=""):
    super(NewFilter, self).__init__(options)
    self.info = "A new query expansion filter"
```

The most important part for the new service is to implement a “run” function.

```
def run(self, query):
```

In this “run” function we must change (expand) the original Query, specifically, the search terms. For instance, we could write a function that inserts “catz” if the user is searching for “cats”. Initially, the “run” function receives a list of search terms in a string:

```
query.search_terms  -> “cats dogs birds”
```

We can add terms to the string

```
query.search_term = query.search_term + “ catz”
```

At the end:

```
query.search_terms  -> “cats dogs birds catz”
```

There are some examples included in the module Search, like the file `TermExpansionFilter.py`. It is really advisable to study this example carefully. In this file, a filter adds some new terms to the original query. The list of new terms is taken from the options previously initialized in the constructor.

6.5 Constructing a query suggestion service

Another type of query filter is the query suggestion filter. In this case, the query terms are unchanged, but the query is decorated with suggestions, a collection of related terms.

The Query object has a property called “suggestions” and the query suggestion service should change it.

The process to add a new query suggestion service is quite similar to the explained in the last section. The new function must derive from `QueryFilter` and it is necessary to add a new constructor and implement a new “run” function that changes the `query.suggestions` field.

There is a trick using `query.suggestion`. As we learn in the last section, `query.search_terms` is a simple sequence or terms. But the suggestion list is a two-dimensional list. It is a list of terms, but each term has its own list of associated terms.

Term	Associated terms
animal	Cat, dog, horse, frog
ship	motorboat, sailboat, merchant
plane	

A “run” function has to take the `query.search_terms` object and add some suggestions according the search items, for instance consulting a dictionary or web service. It is possible also to modify the suggestions from other services.

Usually, the `query.suggestion` field is initially empty.

```
query.suggestion -> “”
```

If we want to add the suggestion described above, the corresponding object in python would be:

```
query.suggestions = {
     u'animal': [u'cat', u'dog', u'horse', u'frog'],
     u'ship': [u'motorboat', u'sailboat', u'merchant'],
     u'plane': []
     }
```

Another example of the format (using python) of these suggestions can be seen in the file `SuggestionFilter.py`

```
query.suggestions = {
     u'school': [u'science', u'reading', u'writing', u'social studies',
     u'arithmetic'],
     u'technology': [],
```

```
    u'news': [u'magazines', u'current'],
    u'puzzles': [],
    u'games':[u'online', u'flash']}
```

6.6 Constructing a results filtering service

Another type of service is the result filtering service. A result filtering service allows processing results contained within the `Response` object. In the source code, these filters are in the Result module.

In this case, we will explain how to develop a new result filter that substitute inadequate words for children.

The structure of a results filter is quite similar to the query filter explained in the last sections. In this case, a new result filter must inherit from the class `ResultFilter`

```
class SuitabilityFilter(ResultFilter):
```

As it happens with the query filter, a new results filter must define a constructor which calls to the base class constructor

```
def __init__(self, options=""):
      super(SuitabilityFilter, self).__init__(options)
      self.info = "New search results filter."
```

The main functionality of the results filter must be in the “run” method. The run method takes a Response object as input and modifies it. The Response object is defined in `OpenSearch.py`. Basically, it contains some general data about the results, and a list of items. An example of input for a results filter can be:

```
self.title              -> 'Yahoo!: Elmo'
self.description        -> 'Search results for “Elmo” at Yahoo!'
self.total_results      -> 10
self.start_index        -> 0
self.items_per_page     -> 10
self.query              -> 'Elmo'
self.items              -> list of response items
self.namespaces         ->{ }
```

If you remember the JSON output format from Yahoo! BOSS, it does not provide a title or description, but `Yahoo.py` adds one.

The most important element of Response is the list of items. An example of item:

```
item.title ->     ‘Home - Sesame Street’
item.link ->      "http:\/\/www.sesamestreet.org\/"
item.description ->”Welcome to Sesame Street! Play educational games
                  and watch videos with <b>Elmo<\/b><b>...<\/b>”
```

Let’s suppose that “Sesame” is an inadequate word for children. A results filter can replace all occurrences of “Sesame” by “Sesamum” in the description.

```
item.description = item.description.replace(“sesame”,”sesamum”)
```

The modified output from the filter will be:

```
item.description ->"Welcome to Sesamum Street! Play educational games
                    and watch videos with <b>Elmo<\/b><b>...<\/b>"
```

We can add a new filter for changing the Response object in any way, for example for translating the results, checking for broken links, or deleting results not suitable to the child's age. There is another example about this functionality in file `SuitabilityFilter.py`. This filter calls a suitability evaluator program and then omits the results not suitable according to a score.

6.7 Summary

The PuppyIR Framework has been created with a modular design in mind, with well defined interfaces and clear separation between modules functionality. It is easy to add new services just adding new files, usually without changing the existent code.

7 Release Plan for v1.0.

Source code for both the prototypes and framework branches have already been uploaded to the project's Sourceforge site[1] and made available for developers to access via Subversion. The plan is to continue the development, documentation and testing of the initial codebase, concentrating on the core components for building search services. The scheduled month for release is M20, and this will be marked by creating a downloadable release with accompanying developer documentation. In more detail:

The following components will be available in v1.0:

- Service Component (puppy.service)
- Model Component (puppy.model)
- Query Component (puppy.query)
- Search Component (puppy.search)
- Result Component (puppy.result)

Work will begin on the following modules that will be included in v2.0:

- Client Component
- Admin Component
- Logging across all components

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] **D2.1 – Collaborative Interactions Models and Interfaces**, PuppyIR, 2010

[5] **D3.1 – Report of Data Pre-processing**, PuppyIR, 2009

[6] **D3.2 – Demonstrator of Query Assistance**, PuppyIR, 2010

[7] **D4.1 – Specification Report**, PuppyIR, 2010

[8] **D4.2 – Design Report**, PuppyIR, 2010

[9] **Python** (Programming Language): http://www.python.org

[10] **Django Project** (Web Application Framework): http://www.djangoproject.com/

[11] **jQuery** (AJAX Library): http://jquery.com

[12] **MT4J** (Multitouch for Java Framework): http://www.mt4j.org

[13] **Open Search Standard - OSS** (Data Format): http://www.opensearch.org/Home

[14] **Open Search Specification**, http://www.opensearch.org/Specifications/OpenSearch/1.1

[15] **Javascript Object Notation - JSON** (Data Format): http://json.org

[16] **UsaProxy** (Logging Webpage Events): http://fnuked.de/usaproxy/

[17] **Log4Py** (Python Logging Framework): https://sourceforge.net/projects/log4py/

[18] **Logbook** (Python Logging Framework): http://packages.python.org/Logbook/

[19] **Rome** (Java Toolkit for Syndicated Feeds (RSS/ATOM)): https://rome.dev.java.net/

[20] **Pymur** (Python / Lemur Integration): https://projects.dbbe.musc.edu/trac/pymur

[21] **Yahoo! Boss homepage**: http://developer.yahoo.com/search/boss/

[22] **Yahoo! Boss Documentation**: http://developer.yahoo.com/search/boss/boss_guide/