IST FP7 231507

D4.5 Report on Specification and Design Changes

File name	PuppyIR_D4.5_Report_on_Specification_and_Design_Changes_v1.0
Author(s)	Richard Glassey (UGLW) Leif Azzopardi (UGLW) José Miguel Garrido (ATOS)
Work package/task	WP4
Document status	Final
Version	1.0
Contractual delivery date	M26
Confidentiality	Public
Keywords	Framework, Design and Specification
Abstract	Summary of the Specification and Design Changes occurring in the PuppyIR Framework since D4.1 Specification Report and D4.2 Design Report

Table of Contents

Executive Summary

This report provides the project with an update on the changes to the specification and design of the PuppyIR open source framework. The experiences gained in the delivery of three prototype services (FiFi, SeSu, and JuSe) and the ongoing development of two project demonstrators (Museum Demo and Hospital Demo) have led to some refinements and simplifications of the initial specification (reported in D4.1 – Specification Report) and overall design (reported in D4.2 – Design Report) of the framework.

This deliverable builds upon previous WP1 and WP4 deliverables:

- D1.2 – Agreed User Requirements and Scenarios **[2]**
- D1.3 – Agreed Technical Requirements **[3]**
- D4.1 – Specification Report **[4]**
- D4.2 – Design Report **[5]**

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 an update of the Implementation status of the framework

2. Outline the changes in Specification of the framework

3. Outline the changes in Design of the framework

This report serves 2 key purposes:

1. Report the significant specification and design changes that have occurred in the framework since the initial deliverables **D4.1** and **D4.2** were submitted.

2. Communicate to all project members and stakeholders how the project is progressing from the initial implementation towards **D4.6 – Release of v2.0** and **D4.7 – Release of v3.0**.

1.1 Implications from D4.4 – Release of Open Source framework

D4.4 provided the project (and wider public) with the first functional implementation of the framework. This release incorporated the specification and design guidelines from **D4.1-Specification Report** and **D4.2 – Design Report** and delivered the core components necessary to build a basic PuppyIR service. These components included wrappers for many online search engine APIs, a common data model for results coming from diverse search engines, and the abstract classes for building PuppyIR query and result filters.

The development experience that went into **D4.4** has provided the team with more insight into how the framework can be improved to simplify the creation of PuppyIR services. Many of these insights have helped to refine the initial design, and remove some of the complexity that did not contribute significantly to the project.

1.2 Structure

The report is divided into the following major sections:

- **Section 2 – Framework Review**
- **Section 3 – Specification Changes**
- **Section 4 – Design Changes**

2 Framework Review

At this stage in the development of the framework, the core components have had an initial implementation, and can be used to create basic PuppyIR services. For example, **JuSe** (Junior Search) is a service that requires video search results and uses the framework to retrieve results from Youtube. It uses the *TermExpansion* query filter to skew search results towards 'cartoons'. Whilst it does not use any of the result processing components at present, a soon to be integrated classifier for filtering by appropriateness will be available. The JuSe service of architecture is shown in the following diagram (greyed out components are unused):

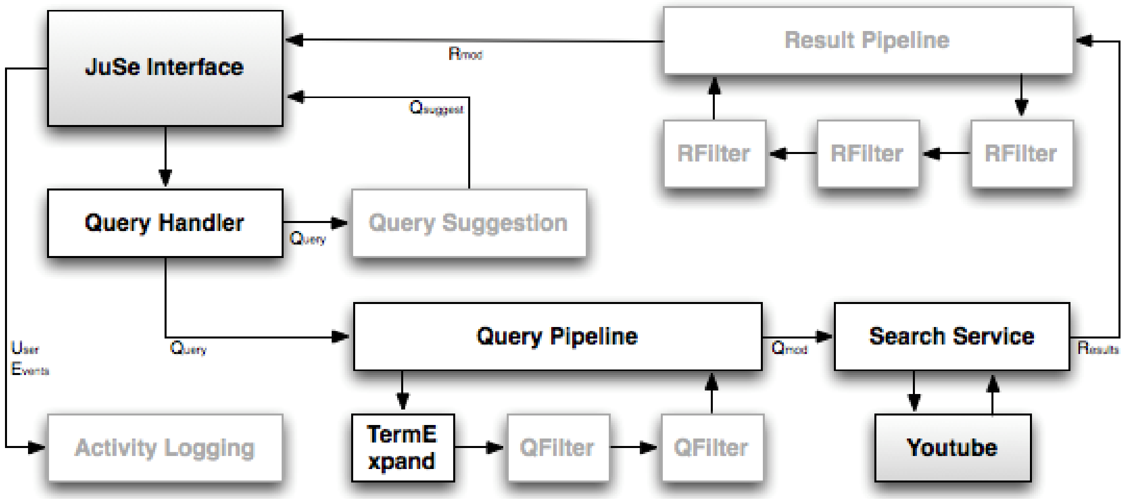

Figure 1: Architecture of JuSe, illustrating active framework components

2.1 Implementation Status

Of the eight major components of the framework, the five most critical (*Service, Model, Search, QueryFilters and ResultsFilters*) have been implemented. The diagram below shows the state of the implementation and the components left to be implemented (**Admin**, **User** and **Client**).

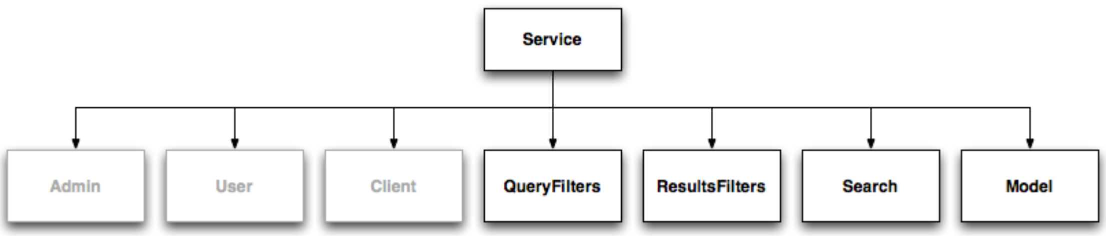

Figure 2: Overview of framework components that have been implemented as of v1.0

The implemented components from the framework allow PuppyIR services to be developed, and also for the results of the research side of the project to be incorporated into the development side. For example, under the *Results* branch of code, an abstract *ResultFilter* object can be extended to create new filters that handle results in a consistent manner. The research on developing suitability measures for unseen content from web pages from **[6]** has manifested itself as a reusable component called the *SuitabilityFilter*. Now any service can make use of this novel component to help tune the level of content that is returned from a PuppyIR search request.

Despite the presence of the initial version of the framework, and the inclusion of novel research, there is still a gap between the current state of the research progress within the PuppyIR project and the development status of the framework – although understandable, this is a matter that will be addressed in the forthcoming versions of the framework.

2.2 Implementation Roadmap

The forthcoming release of the framework (**D4.6 – Release of Framework v2.0**) will incorporate the following major features, associated with the components not yet implemented:

- **PuppyIR Service Configurator –** create a standalone and Django-based administrative web interface that will provide a user friendly means of creating, configuring and evaluating PuppyIR services (part of *Admin* Component).

- **Basic User Profiles –** use the core Django user components to provide basic user profiles that will allow authentication and personalisation (part of *User* Component).

- **Activity Logging Tool -** to complement the standard query logging functionality, client interfaces that are web-based will also be able to log the user activity events. This level of information can be configured and stored by the server for analysis (part of *Client* Component).

- **Component Integration –** As mentioned in the previous section there will be an ongoing effort (through versions 2.0 and 3.0) to ensure that the research that has been produced makes it into the framework in the form of reusable components that PuppyIR services can employ (part of *Query* and *Results* components).

3 Specification Changes

Deliverables D4.1 – Specification Report and D4.2 – Design Report were delivered in the early stages of the project, contractual delivery dates months 10 and 2 respectively. Despite appearing before the majority of work in the project had started, the specification and design were designed with flexibility in mind. This has had the advantage that at this stage, going into the final year, the scale of changes is not overwhelming, nor has any major redesign been necessary.

In part, this has been achieved by going beyond the contractual obligations and developing a variety of prototypes (FiFi, SeSu, and JuSe – see **D4.3** and **D4.4** for more information about the aims and implementation of these prototypes) that have helped to inform major decisions and avoid pursuing ill-thought out ideas.

However, with experience of the first two years, there have been some changes to the original design and specification. This section briefly outlines the changes to the specification and provides a justification for these decisions.

3.1 Simplification of Access Channel

D4.1 (Section 2.3, page 7) discussed three major access channels: Desktop PC Devices, Multi-touch Devices and Tangible Devices. One major change here was the decision not to make a specific client-side component to access a PuppyIR service. This was envisaged to be a simple application programmer's library for Touch and Tangible Devices. Whilst this may have made development easier for a particular platform, we would have limited our selves to a single platform, e.g. a Java component to be incorporated into a Java-based multi-touch library such as MT4J. In reality, the simpler option and more widely compatible approach is to only use the REST-based interface to a PuppyIR service, as described in **D4.2**. As this was already a major part of the framework, no extra effort is required. Also, most development platforms have a library for constructing, sending and receiving HTTP messages over a network. This method also is equally valid for services that are completely independent and only communicate locally within a single host.

3.2 Simplification of Local Search Integration

One of the useful features provided within the PuppyIR framework is the unification of results from multiple search engines (e.g. Yahoo, Youtube, Twitter, etc). Within the prototypes developed so far, both local and remote search engines have been used. Whilst remote (or online) search engine APIs are relatively easy to integrate into the framework, local search engines have not all been as straightforward to integrate due to the interface inconsistencies and less of a focus on providing web service access.

Local search engines, such as Lemur and Terrier, are primarily research projects. As a result, they have not focused much effort in creating a stable and simple to use public API. This restricts their usage to certain platforms based upon their language of implementation, or involves depending upon an independent wrapper library. For example, in the case of the first prototype – FiFi, a library called Pymur (implemented in Python) was used to interact with Lemur (implemented in C/C++). Although this is satisfactory for a prototype service, it would mean that including it within the framework would require updates to Pymur to be incorporated as and when they happened. Therefore a decision was taken to focus on search engines that have well maintained APIs, that use the REST paradigm to ensure platform independence. Examples here are PF/Tijah and Solr/Lucene.

3.3 Simplification of Logging Activity

In line with Section 3.1, the development of specific components for Multi-touch and Tangible Devices has been abandoned. The main focus instead is to log queries for all requests from all access channels, and in the cases where the service is web-based, the intention is to develop an activity logging tool that can capture multiple levels of granularity, ranging from coarse grained page elements to finer grained mouse movements and positions on the page.

Part of the motivation for this is due to platform consistency. With web-based services it is possible to create a single logging tool in JavaScript that will capture and log user interaction events occurring in the web browser. It is much more difficult to anticipate logging needs on more specific platforms such as touch-tables with custom-built applications, i.e. non-web browser based. One opportunity however is to investigate how far the web-based activity logging also works on web applications running on multi-touch devices.

4 Design Changes

The following sections discuss the changes that have occurred in the design of the PuppyIR framework since the submission of **D4.2 – Design Report**.

4.1 Separation of Search Component

Section 4 of **D4.2** presented the Composition Viewpoint, a hierarchical breakdown of the major components of the framework. Figure 2, in Sec. 2.1, shows the current state of this viewpoint. The major change has been a reorganisation of the responsibilities of the *Query* and *Results* components. Both of these components had classes to handle the interaction with search engines. It became apparent when revisiting this design idea during early implementation that it would be much clearer if the search-concerns of the framework were better separated, especially given the importance of search within the framework. This can be illustrated by comparing the Logical Viewpoint (class-level) diagram's for both the *Query* (see Fig 5 – **D4.2**) and *Results* Components (see Fig 6 – **D4.2**), as shown in Fig. 3 below.

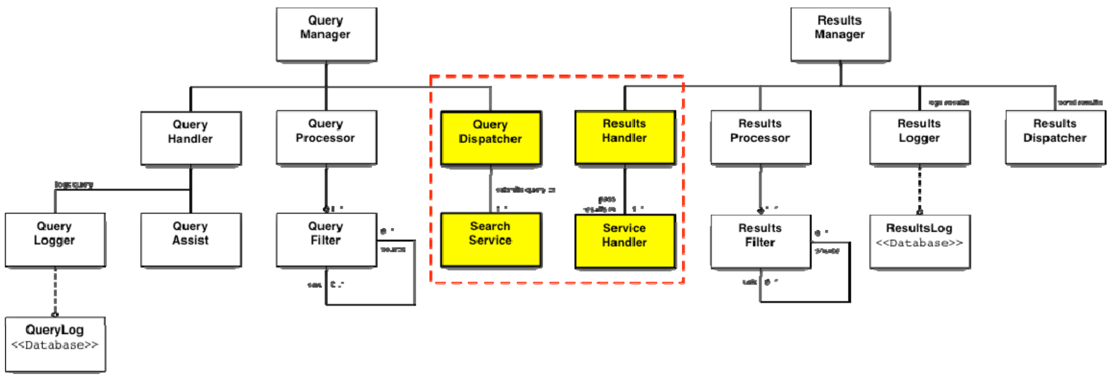

Figure 3: Candidate components for new Search component

4.2 Standalone Mode

Another contribution from Sec. 4 of **D4.2** was the recommendation that web-application frameworks would be an ideal fit for the needs of implementing the framework. The main advantages here would be the ready availability of components to satisfy some of the framework requirements in the key areas of handling HTTP communication, providing a REST interface, object-to-relation mapping and general user administration and authentication. However, it became clear during development that there were many advantages to keeping much of the framework separate from a specific web-application framework.

To illustrate, during development many simple services were created to test the search services, query and result pipelines and various other configurations of core components. Often, there was little need to build a web interface to the service for remote access. Therefore, whilst PuppyIR builds on the Django web-application framework **[7]** to create PuppyIR services, the framework can be used separately for testing and experimental work. Furthermore, should Django cease to be supported and developed, the separation should ensure that it would take little effort to integrate the framework into another compatible platform.

4.3 Content Management Tasks

The current version of the framework does not directly support content management. This is a feature that can be built using the Django web-application framework. Django excels at building content management solutions and does not require any specific features from the framework. This does entail that whenever a service is built that has a local collection of data that is evolving, part of the development effort will need to include building in content management support. Whilst this shifts effort towards the service developers, they will have more understanding of the data schema needs for a particular service, and increasingly search engines such as Solr/Lucene have web-application framework integration that indexes content (e.g. Project Haystack **[8]** provides integration between several popular local search engines and Django).

4.4 Configuration Management

Part of the *Admin* component, the *Service Configuration*, is used to configure the search service, query/result processing and logging. In **D4.2** it was envisaged that there would be a single point of configuration. However, with the standalone mode (see Sec. 4.2) and web-service mode, a different approach to configuration is required. The main change here to the design is that the *Service Configuration* component checks for the presence of a database model for managing the configuration of a web-service. If this exists, then the configuration is chosen from the database, rather than from a static file. The reason for the database model of configuration is that Django provides a default admin interface that creates a user friendly and dynamic method of changing the service configuration, rather than having to manually edit the code.

4.5 User Model and Authentication

As in the previous sections, user modelling and authentication is delegated to the web-application framework. In the same sense as providing content management facilities, it is difficult to anticipate the user modelling needs of PuppyIR services. However, Django provides a straightforward system for managing users, authentication and handling sessions.

4.6 Query Assist / Search Suggestion

Section 5.3 of **D4.2** outlined the *Query* Component of the framework (see Fig. 3). In the original design, each query processing pipeline could have an optional *Query Assist* service attached to it. However, it became clear that the *Query Assist* process itself is just another version of the query processing pipeline. Therefore, if a service needs query suggestion, a separate query processing pipeline can be constructed to facilitate this. This change has the benefit that now the query suggestion itself can be altered using both query and result filters, as shown in Fig. 4.

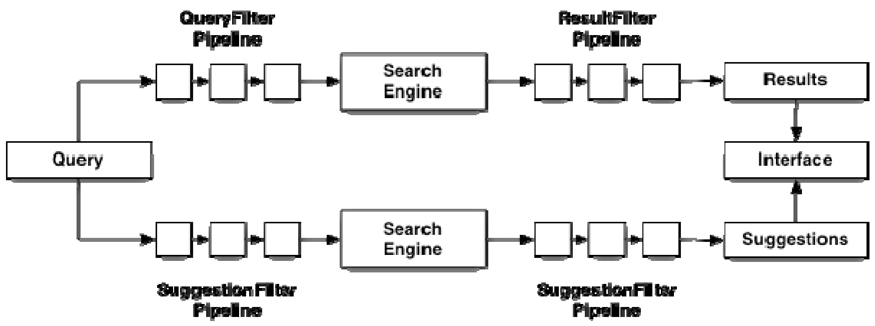

Figure 4: Query suggestion reuses the query/search/result pipeline

5 Summary

This report has reviewed the major deviations from the original specification and design of the framework. Both **D4.1 – Specification Report** and **D4.2 – Design Report** were submitted within the first year of the project. They were based mainly upon the overall Project description, **D1.2 – Agreed User Requirements and Scenarios**, and **D1.3 – Agreed Technical Requirements**. Since these early stages, the project development team have had continuous experience with developing prototypes and specifying project demonstrators. This experience has provided the opportunity to revisit and revise some of the specification and design ideas so that the framework is better able to build useful, realistic PuppyIR services.

6 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] **D4.1 – Specification Report**, PuppyIR, 2010

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

[6] **"Web Page Classification on Child Suitability",** Carsten Eickhoff, Pavel Serdyukov, Arjen P. de Vries. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM), 2010.

[7] **Django Web Application Framework Homepage** – https://www.djangoproject.com/

[8] **Project Haystack Homepage** – http://code.google.com/p/django-haystack/