D4.2 Design Report

File name	PuppyIR-D4.2-DesignReport-1.0.doc
Author(s)	Richard Glassey (UGLW) Leif Azzopardi (UGLW) José Miguel Garrido (ATOS) Jeldrik Schmuch (ATOS)
Work package/task	WP4
Document status	final
Version	1.0
Contractual delivery date	31 March 2010
Confidentiality	confidential
Keywords	Design and Implementation
Abstract	This report details the design of the open source PuppyIR framework. The design is derived from the deliverables D1.2 (User Requirements and Scenarios), D1.3 (Agreed Technical Requirements), D3.1 (Report on Data Pre-processing), D4.1 (Specification Report) and the Report on Implementation of Prototypes and Demonstrators (under construction). The purpose of this report is to document the software design description (SDD) that will guide the development and release of the Open Source PuppyIR framework.

Table of Contents

Executive Summary

This report details the design of the open source PuppyIR Framework. This design is derived from the following deliverables from WP1, WP3 and WP4, and from the plan for the work package aiming at the implementation of prototypes and demonstrators (final version description due end of May 2010).

- D1.2 – Agreed User Requirements and Scenarios
- D1.3 – Agreed Technical Requirements
- D3.1 – Report on Data Pre-processing
- D4.1 – Specification Report
- Implementation of Prototypes and Demonstrators (WP7)

The purpose of this report is to document the **software design description (SDD)** that will guide the **development** of the **PuppyIR framework** and the Year 2 deliverables D4.3 - Report on Implementation and Documentation and D4.4 - First Release of Open Source framework.

The software design description presented here extends from D4.1 – Specification Report [5], which described the overall software architecture of the framework. The purpose here is to describe in detail the design to guide the implementation choices and development of the framework. This will be achieved by describing the design using a series of design views to represent the design concerns identified in D1.2 – Agreed User Requirements and Scenarios [2], D1.3 – Agreed Technical Requirements [3] and D3.1 – Report on Data Pre-processing [4]. Each design view will use an appropriate **design viewpoint** with a corresponding set of **design elements** (**entities**, **attributes**, **relationships** and **constraints**) that will be described using the relevant **design language** (e.g. Unified Modelling Language, Entity-Relation Diagram, etc). The following design viewpoints of the PuppyIR Framework are described:

- **Composition Viewpoint**
- **Logical Viewpoint**
- **Information Viewpoint**
- **Interface Viewpoint**
- **Interaction Viewpoint**

The report concludes with a discussion about the design rationale and a tentative plan for initial implementation of the 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]. 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 the design of the PuppyIR Framework (henceforth referred to as the 'framework'). The goals of this report are to:

1) Describe the design of the framework from multiple viewpoints,
2) Make technology recommendations for implementation of the framework
3) Explain the overall design rationale
4) Outline the initial plan for implementation

1.1 Audience and Scope of Design

This report serves two key purposes:

1. Provide a software design description document to members of the project responsible for the software development (the developers) of the framework to help guide the development effort throughout the lifespan of the project.

2. Communicate to all project members and stakeholders how the project is progressing from the initial requirements and specification reports towards an open source framework that will facilitate the construction of child-friendly information services.

The scope of the report is limited to software design description, technology recommendations and design rationale of the framework. Detailed implementation decisions are delegated to work to be described in the forthcoming D4.3 – Report on Implementation and Documentation, which will build upon and update the software design description presented in this report, providing additional details on how the framework itself has been implemented, based upon experience gathered in producing project demonstrators.

1.2 Implications from D4.1 and WP7

In D4.1 – Specification Report a high level analysis of the system context of a PuppyIR 'Service' and architecture for the framework was described. This report continues by adding detail to the high-level view by using a variety of design views and design languages with appropriate commentary.

Several of the components identified in D4.1, in particular within the sections concerning the functional (D4.1 – Sec. 4.2) and physical (D4.1 – Sec. 4.3) architecture, form the basis of the initial design views discussed. In some cases, names of components have been altered for simplification and better clarity of design. However, there have been no major deviations or additions in the design vision described within D4.1.

In the interim period between the delivery of D4.1 and production of this report there has be a ongoing effort within WP4 to develop initial prototypes to help inform the design and implementation decisions going into the framework. This has helped to reveal design concerns as well as useful technologies and standards that the design of the framework will take into consideration and potentially incorporate. This investigation and design revision will continue on an ongoing basis along with the implementation of project demonstrators.

1.3 Terminology

The following standard terms and definitions are used in this report [6]:

Term	Definition
Design Concern	An area of interest with respect to a software design.
Design View	A representation comprised of one or more design elements to address a set of design concerns from a specified design viewpoint.
Design Viewpoint	The specification of the elements and conventions available for constructing and using a design view.
Design Element	An item occurring in a design view that may be any of the following: design entity, design relationship, design attribute, or design constraint.
Design Pattern	A general reusable solution to a commonly occurring problem in software design, consisting of a description or template for how to solve the problem.
Design Language	A standardised means of communicating the concepts of a particular software design using either graphical or textual descriptions with well-defined syntax and semantics.

1.4 Structure

The design report is divided into the following major sections:

- **Section 2 – Architecture Review:** A reminder of the framework architecture from D4.1
- **Section 3 – Design Approach:** A brief discussion regarding the approach taken in describing the design for the framework; introducing the five design viewpoints.
- **Section 4 – Composition Viewpoint:** describes the framework from the viewpoint of the composition of modular components using the UML Component Diagram.
- **Section 5 – Logical Viewpoint:** describes the static structure of each of the components introduced in the previous section using the UML Class Diagram.
- **Section 6 – Information Viewpoint:** describes the persistent information that will be required by the framework using the Entity-Relation Diagram.
- **Section 7 – Interface Viewpoint:** describes the high-level interfaces that each component exposes or depends upon using the UML Component Diagram.
- **Section 8 – Interaction Viewpoint:** describes the sequence of communication between classes within and between components and across device boundaries using the UML Sequence Diagram.
- **Section 9 – Design Rationale:** a brief discussion justifying the approach taken within this report and an identification of the areas for revision.
- **Section 10 – Implementation Plan:** an outline of the initial plan for implementation of the framework with reference to future deliverables.

2 Design Approach

An over-arching goal of this design report is to allow for implementation flexibility and to avoid over-engineering the framework design. It is expected that this report may undergo iterative revision as experience is gathered through the implementation of the framework and associated demonstrators. Therefore, the following sections seek to best illustrate the abstract design elements of the framework and their logical composition and interaction. It may well be the case that a one-to-one mapping between the design elements presented here and the actual implementation of the framework requires some extrapolation (or indeed is already provided for by an existing library or framework), however for this initial report, clarity of design is preferred over comprehensive coverage of every last detail.

One of the key contributions of this report will be the identification and description of the hot-spots (areas that can be extended to customise a PuppyIR service, e.g. how queries are processed) and the frozen-spots (areas that all PuppyIR services should have in common, e.g. administrative access to service configuration) in the framework. Ultimately these are the significant features of the framework that will influence how successful it is in developing multiple and distinct information services for children.

Where possible, based on the experiences of developing prototypes and prior implementation experience, recommendations will be made for libraries and frameworks that might be reused within the development of the framework. As there are several large areas of functionality, such as handling HTTP communication, processing XML, logging events and mapping objects to relational databases, which have well established solutions, it is expected that development effort should be made more efficient by the reuse of appropriate open source software.

This design report follows the recommendations laid out in the IEEE Std for Software Design Description [6] and decomposes the high-level architecture provided in D4.1 into a set of design views and associated design viewpoints. Each design view seeks to address a particular set of design concerns (e.g. how is persistent information modelled in the framework). The view is then governed in what it communicates by adopting a particular viewpoint and using a design language and associated commentary to describe the key points of the view.

Five **design viewpoints** have been selected as relevant to the design of the framework: Composition, Logical, Information, Interface, and Interaction. The IEEE Std for Software Design Description lists another seven viewpoints (Context, Dependency, Patterns, Structure, State Dynamics, Algorithm and Resources) which were deemed unnecessary, or already adequately described via the selected viewpoints. 1) Context - D4.1 already describes the system context in terms of the users (and their use cases), channels and external entities that are involved with an implemented service. 2) Dependency, Structure and Resources – The Interface Viewpoint (see Sec. 7) includes the structure and dependencies of components within the framework as well as the dependencies on external resources (databases and search services). 3) Patterns – Throughout the discussion of each viewpoint, particular design patterns will be identified and referenced if considered relevant. 4) State Dynamics and Algorithms – These are considered to be too low level for consideration at this stage of design.

Throughout this report the primary **design languages** used are the Unified Modelling Language (UML) [7] and the Entity-Relation Diagram (ERD) [8]. They have been chosen because of their wide acceptance in the field of software engineering and efficacy in communicating software design concepts between developers. (although we accept they do have their weaknesses [9]) Their level of abstraction is suitable as it is possible to illustrate structure, behaviour, interaction and data models compactly using a clear graphical notation that is widely used and understood by software developers. Where extra detail and explanation is required, each diagram is accompanied by a short textual description highlighting the key concepts.

3 Architecture Review

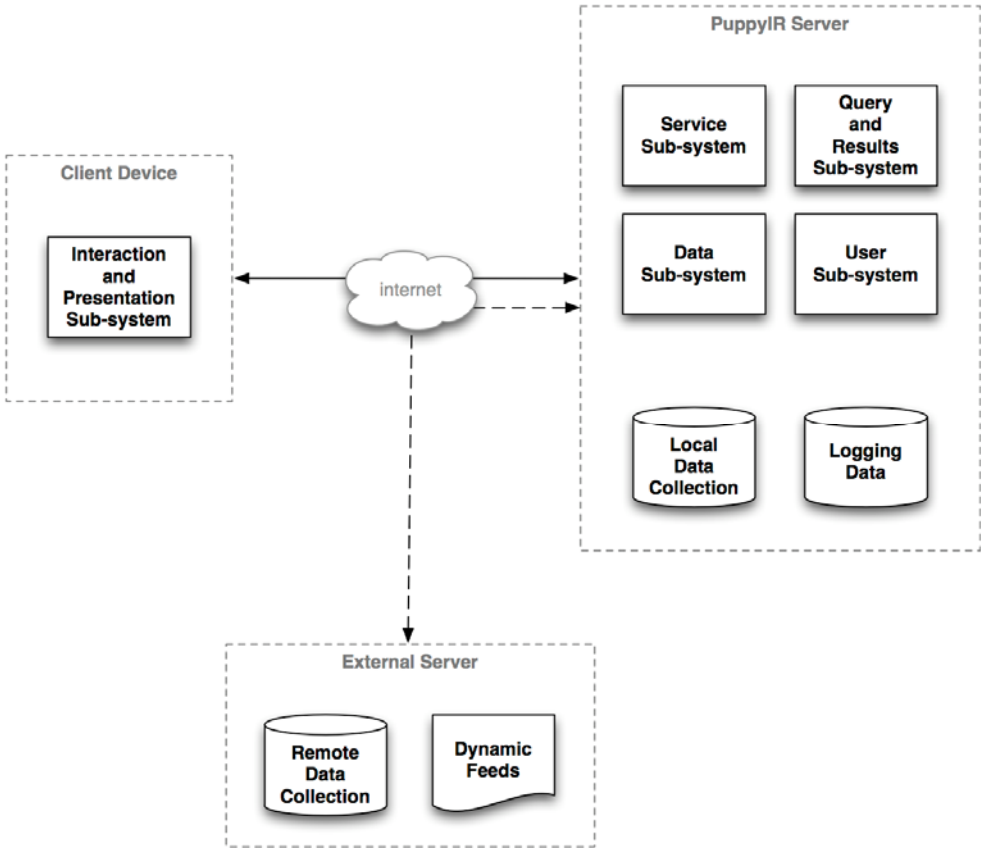

Figure 1: Functional and Operational Architecture of framework (see D4.1)

Figure 1 shows how the functional components of the framework are to be deployed across a physical system of devices and networks. The main observations are that a PuppyIR service acts as a mediator between the client device of the user (handling the interaction and presentation concerns) and local and remote services (or more simply, 'search services') that allow a particular data collection to be queried.

This software architecture [10] reflects the aim of the project in simplifying the creation of search services that are tailored to children's needs, rather than building a new search service from first principles. The result is a framework that can act as an enabling platform for the production of multiple, distinct information services for children that can be accessed from multiple devices and make use of multiple collections of data.

The collections of views that follow can be seen as an extension of the original views taken in the previous report (system context, high-level architecture, functional and operational architecture [11]), each contributing more detail to the design solution, such as identifying specific design patterns [12] to use and specific classes and their relationships. As mentioned, there are several minor changes from the terminology used in Fig. 1, which discusses the expected sub-systems of the framework, and the components described here that are to be implemented as part of the framework. The association should be clear from the context of the description.

The following sections of this report will address a variety of design concerns that attempt to produce a coherent software design description that will achieve the high-level vision of the architecture described originally in D4.1 – Specification Report.

4 Composition Viewpoint

Viewpoint Description

The composition viewpoint describes how the framework is hierarchically structured into its constituent components, as shown in Figure 2. The following sections address the design concern and describe the design elements involved and any implementation notes.

Design Concern

Identify the composition and modular assembly of major components in the framework.

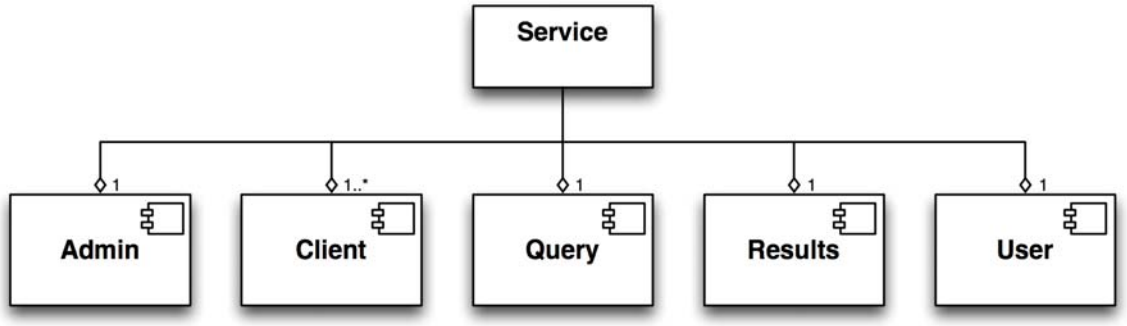

Figure 2: Composition viewpoint illustrating the major components of the framework

Design Elements

Service

- Type: System
- Description: The **Service** component in this diagram represents the global system, one that has been created from the framework. It consists of five major components: **Admin**, **Client**, **Query**, **Results**, and **User**. As shown in the previous section, the components are not all physically co-located on a **single server**. Instead, the **Client** component of the framework consists of the software that will be deployed upon the physical **client device** that a user interacts with. Of particular note at this stage is the indication that there can be one or more Client components. This reflects the fact that not all Client Devices will use identical implementations of the Client component. For example, the Client component for a multi-touch table (e.g. integrated with a multi-touch framework like MT4J [13]) will differ from the Client component of a web application (e.g. implemented with an AJAX library such as jQuery [14]). That said, their functionality will be equivalent. The four remaining components are collocated on the same server.

Admin

- Type: Component
- Description: The **Admin** component is responsible for configuring the service (e.g. which search services to dispatch queries to) and managing registration and authentication of the users of the system.

Client

- Type: Component
- Description: The **Client** component is responsible for communicating with the service (sending queries/interaction events and receiving results/information), storing the access credentials of the user (to minimise repeated authentication by user) and presenting the user interface to the user. This functionality is expected as core to all clients, irrespective

of the device they execute upon. By conforming to the interfaces of this component, a single service can be developed that can support many different clients. In terms of the framework this is the first *region of flexibility* that is fundamental to the aims of the project as explored in WP2 – Interaction Models Design [1]. The ability to explore multiple novel and diverse user interfaces that children may benefit from using when performing information seeking tasks is crucial, and hence the framework reflects this by providing a hot-spot for extending and building user interfaces, rather than deliver a single fixed interface.

Query

- Type: Component
- Description: The **Query** component is responsible for handling the user requests from the **Client**. Primarily this will be in the form of queries sent from the user interface, but extends to other forms of user interaction. This component allows queries to be pre-processed before dispatching them to one or more search services, logged for scientific data-mining and analysis, and provide an opportunity to assist the user in the query formulation process before submission. The query pre-processing is designed to be pluggable, such that a variety of query filtering components can be developed, integrated and potentially chained together in sequence. This is the second *region of flexibility* that the framework exposes in order to explore query handling and to support the development of child-appropriate components that can better understand the intent of the user who may have difficulty in formulating a query due to their limited cognitive development. The simplest example would be a query filter that corrects common spelling mistakes that children tend to make.

Results

- Type: Component
- Description: The **Results** component is analogous to the **Query** component, and is responsible for handling the result sets that are returned whenever a query has be dispatched to a search service. This component handles the incoming results from search services, unifies the representation of search results (e.g. using the Open Search standard for search result representation [15] and an established library to process them, such as Rome (Java) [16]), processes the result set using a similar pluggable pipeline of components used in query pre-processing, and logs them for analysis. This is turn is the third *region of flexibility* of the framework to allow the development of novel components for analysing, moderating, managing and filtering the results that children's search queries return from various search services. A simple example would be a filter that checks the URI of a particular result against a store of known URIs to block. This filter can be developed independently of the rest of the framework yet integrated easily by conforming to the interface of the pipeline.

User

- Type: Component
- Description: The final major component is the **User** component. This is primarily included to build a basic profile of the users, to aid understanding of the interaction challenges that children may face using an information access service. It is not mandatory to build a personal profile for each service, but the capability to do so is included as a core feature. It is understood that users may in fact be groups of children using a collaborative search interface, such as a multi-touch table, therefore collections of users can be represented as groups to help build collaborative services.

Implementation Technologies

Implementing the entire framework from first principles would be prohibitive and not achievable in the time-frame of the project and given the available man-power. Taking these constraints into

consideration, every opportunity will be taken to explore and potentially use existing open source libraries and frameworks.

The first major opportunity for reuse comes when considering the overall construction of a service. It is clear that the recent developments in web application frameworks (WAF) can be leveraged to solve several core issues, such as wiring the components together (integration) and interacting with the clients on remote devices (communicating). A pragmatic choice therefore would be to adopt a suitable WAF because of their core support for:

- handling HTTP communication and providing REST interfaces [17]
- ORM (Object to Relation Mapping) with relational databases [18]
- user account management
- logging integration
- principles such as Inversion of Control / Dependency Injection [19]
- patterns such as Model View Controller [12]

Key Examples of frameworks to build upon:

- **Grails** (Java/Groovy) [20]
- **Django** (Python) [21]
- **Rails** (Ruby) [22]
- **Lift** (Scala) [23]

The second major opportunity for reuse concerns the implementation of clients. As mentioned there will be multiple types of devices to be supported, and it is sensible to integrate the Client component with frameworks like **MT4J** and libraries like **jQuery**. To illustrate, the Client component implemented in jQuery could then be integrated into a larger web application interface designed by a developer. This web application could access a PuppyIR service using its integrated Client component.

5 Logical Viewpoint

Viewpoint Description

The purpose of the logical viewpoint is to elaborate upon the components described in the Composition Viewpoint by describing the static structure and relations of the classes that make up each component.

Design Concern

Identify appropriate abstractions at the class and datatype level that need to be designed for each component of the framework

Design Elements

The design elements for the logical viewpoint are structured according to the component they belong to. The following sections (Sec. 5.1 – 5.5) take each framework component in turn and identifies and describes the constituent design elements. Each component is decomposed and illustrated using a UML Class Diagram (Fig. 3 – 7).

5.1 Admin Component

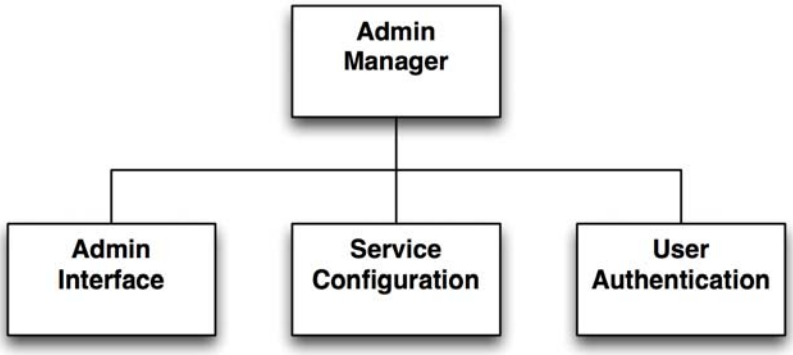

Figure 3: Class diagram of Admin Component

Design Elements

AdminManager

- Type: Class
- Description: The **AdminManager** acts as a simplified interface to the rest of the functionality of the **Admin** component, exposing the major public interfaces, whilst hiding the implementation of the constituent classes. It is an implementation of the **Façade Design Pattern** [12].

AdminInterface

- Type: Class
- Description: The **AdminInterface** is responsible for providing a user interface that allows a system administrator with administrative credentials to access and modify the settings that govern the configuration of the service and also the management of user accounts (i.e. the creation and deletion thereof).

ServiceConfiguration

- Type: Class
- Description: The **ServiceConfiguration** is responsible for managing the specific details of the service, namely the designation of search services to query, the steps in the query processing pipeline, and the steps in the results processing pipeline. Due to its requirement to expose these configurations globally to relevant components, it is an implementation of the **Singleton Design Pattern** [12].

UserAuthentication

- Type: Class
- Description: The **UserAuthentication** is responsible for verifying the credentials of users attempting to log into a service, and establishing a session upon successful authentication.

5.2 Client Component

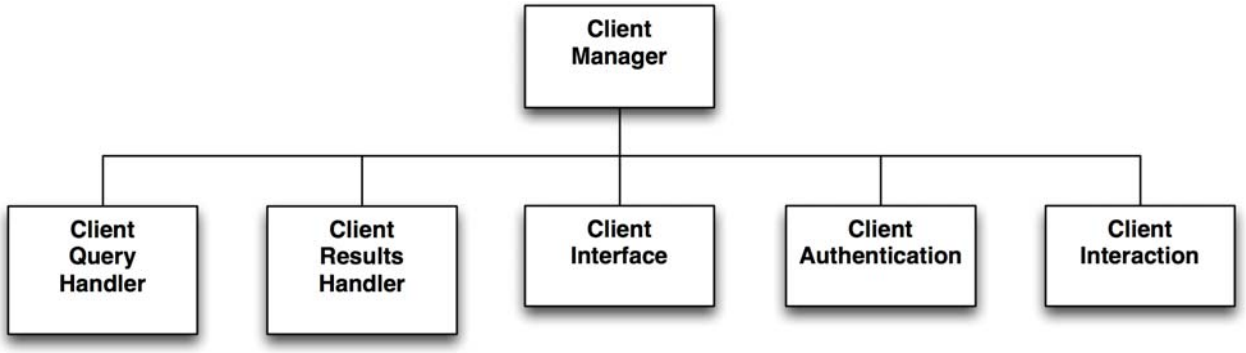

Figure 4: Class diagram of Client Component

Design Elements

ClientManager

- Type: Class
- Description: The **ClientManager** acts as a simplified interface to the rest of the functionality of the **Client** component, exposing the major public interfaces, whilst hiding the implementation of the constituent classes. It is an implementation of the **Façade Design Pattern** [12].

ClientQueryHandler

- Type: Class
- Description: The **ClientQueryHandler** is responsible for capturing the user's query (both in progress of being formed and actually being submitted, e.g. to permit asynchronous communication with the server to offer query assistance, and communicating it to the server in the appropriate format that the server expects. It is an implementation of the **Observer Design Pattern** [12] that observes and responds to events that are occurring within the **ClientInterface**.

ClientResultsHandler

- Type: Class
- Description: The **ClientResultsHandler** is responsible for handling the results that are returned for a particular query. In particular it creates a object representation for the search results which are formatted according to the **Open Search** standard [15] for

results from a search service. This 'model' can then be exposed to the **ClientInterface** in order to update its 'view'.

ClientInterface

- Type: Class
- Description: The **ClientInterface** is responsible for rendering the user interface, capturing the user interaction when formulating queries and interacting with results when returned. As mentioned, the **Client** component is intended to be flexible enough to provide the core functionality to interact with a service, yet delegate the issues of user interface design to the 3rd party developers. In light of this, this class acts as a placeholder.

ClientAuthentication

- Type: Class
- Description: The **ClientAuthentication** is responsible for securely communicating the user credentials to the server to be authenticated, then managing any subsequent session information handling.

ClientInteraction

- Type: Class
- Description: The **ClientInteraction** is responsible for transmitting user interface events to the server-side to be logged for analysis

5.3 Query Component

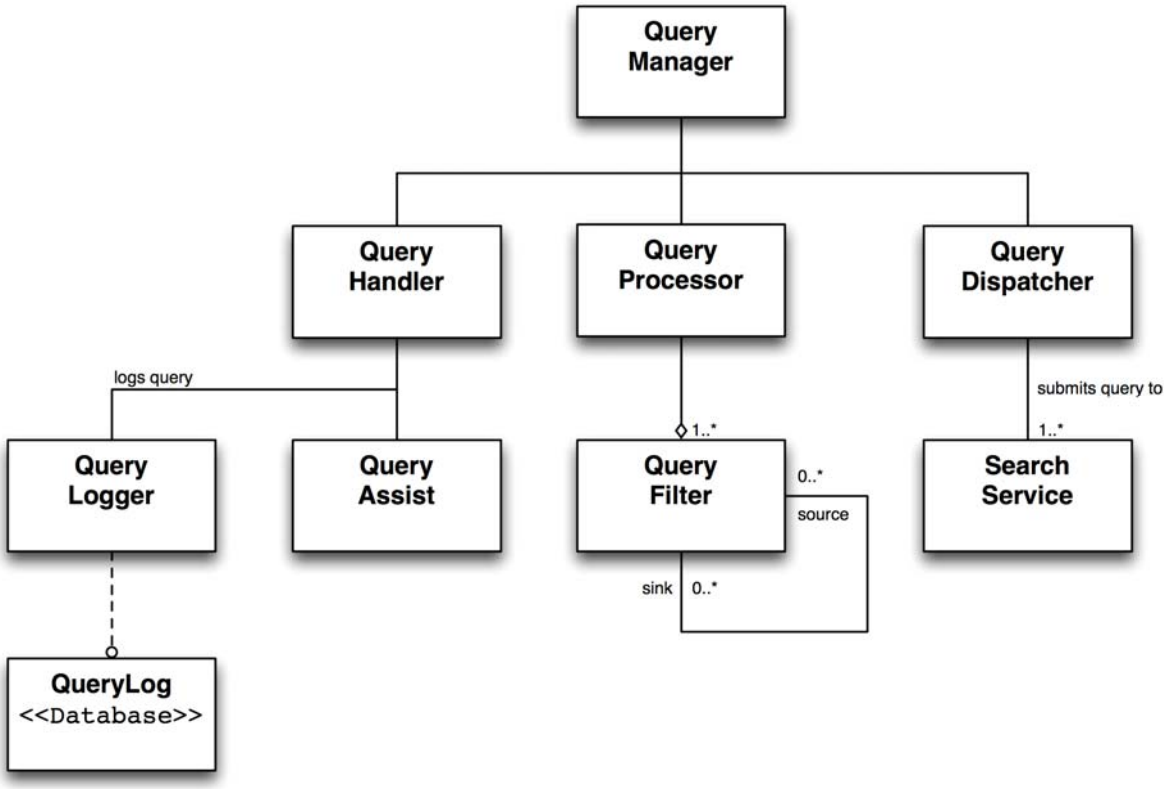

Figure 5: Class diagram for Query Component

Design Elements

QueryManager
- Type: Class
- Description: The **QueryManager** acts as a simplified interface to the rest of the functionality of the **Query** component, exposing the major public interfaces, whilst hiding the implementation of the constituent classes. It is an implementation of the **Façade Design Pattern** [12].

QueryHandler
- Type: Class
- Description: The **QueryHandler** is responsible for receiving the query from the Client component. At the stage of query formulation, there is an opportunity to asynchronously respond to a partial query by involving the **QueryAssist** class. Also, the query can be logged into a datastore using the **QueryLogger**. Once the full query has been submitted, the query is passed onto the **QueryProcessor**.

QueryAssist
- Type: Class
- Description: The **QueryAssist** is responsible for taking incomplete queries and responding to the user with a set of query suggestions that should help with completion of a query that can be displayed by the **Client**. It is expected that one or more concrete instances of **QueryAssist** be implemented in the pursuit of investigating what types of query assistance children respond to and benefits from, whilst allowing unhelpful types to be identified.

QueryLogger
- Type: Class
- Description: The **QueryLogger** is responsible for logging the original and processed queries (see below) to a datastore for later retrieval and analysis.

QueryProcessor
- Type: Class
- Description: The **QueryProcessor** is responsible for managing the query pre-processing that has been specified by the **ServiceConfiguration**. It receives the complete query from the **QueryHandler** and then passes this onto one or more **QueryFilter** components (which form a serial pipeline). The combination of **QueryProcessor** and **QueryFilter** are an implementation of the **Chain of Responsibility Design Pattern** [12].

QueryFilter
- Type: Class
- Description: The **QueryFilter** is responsible for modifying the query in a specific manner. This can be seen as an implementation of the **Template Design Pattern** [12], where the behaviour of handling the query is consistent, but the actual method of modification varies from filter to filter. Each filter takes as input the representation of a query, and simply outputs the altered query using the same representation after processing. The design principle of **Inversion of Control / Dependency Injection** [19] is relevant here, where one or more filters can be specified and wired together via an external source of configuration (e.g. by using the **ServiceConfiguration** class), rather than needing filters to explicitly reference each other.

QueryDispatcher
- Type: Class
- Description: The **QueryDispatcher** is responsible for sending the complete and (potentially) processed query from the **QueryProcessor** to one or more search services.

It also ensures that for each search service used, there is a corresponding **ServiceHandler** to handle the incoming results.

SearchService

- Type: Class
- Description: The **SearchService** is responsible for representing the interface to a search service. Due to the variety of interfaces to available search services (local and remote systems) this will be an implementation of the **Adapter/Wrapper Design Pattern** [12], allowing 3rd party developers to incorporate new search services by implementing this abstract class. For each search service (or standard such as **OpenSearch** [15]) there will be a corresponding **ServiceHandler** that will handle the results returned.

Implementation Technologies

Crucial to the **Query** component will be the approach to implementing a flexible pipeline. Several frameworks, such as **Spring** (Java) [24], employ the design principle of **Inversion of Control** to provide a more flexible and elegant approach to wiring components together. This often reduces configuring components together to writing configuration files, indicating how components will interact during runtime. Combining this approach with the **Chain of Responsibility** pattern allows a simple lightweight query component pipeline to be configured. This would remove the need for 3rd party developers to understand how to integrate their components into an existing pipeline.

5.4 Results Component

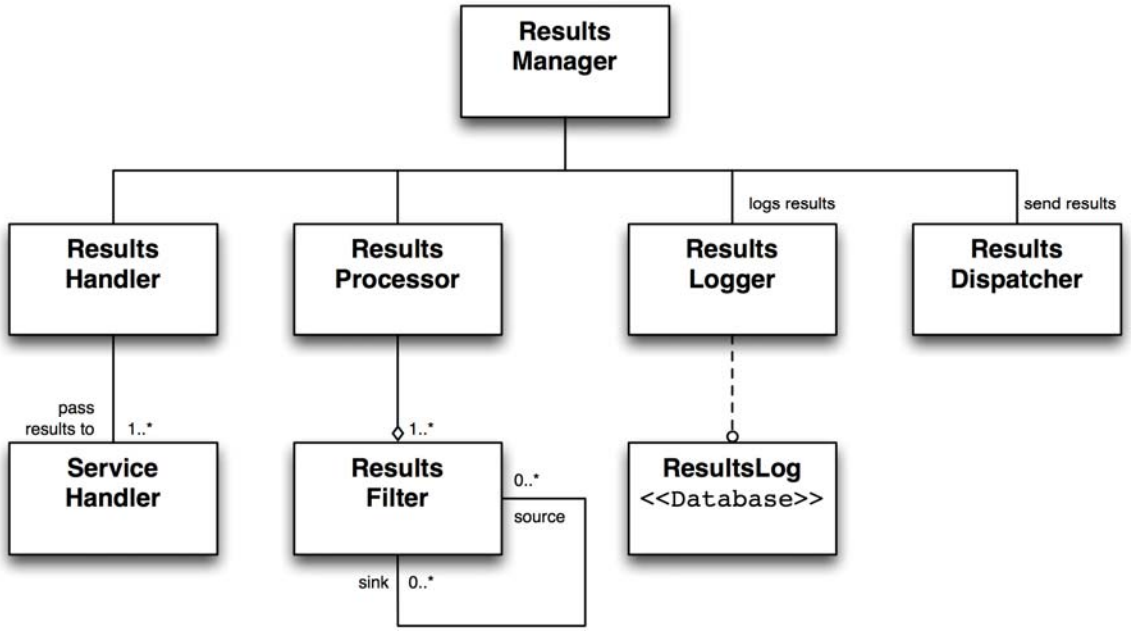

Figure 6: Class diagram for Results Component

Design Elements

ResultsManager

- Type: Class
- Description: The **ResultsManager** acts as a simplified interface to the rest of the functionality of the **Results** component, exposing the major public interfaces, whilst

hiding the implementation of the constituent classes. It is an implementation of the **Façade Design Pattern** [12].

ResultsHandler
- Type: Class
- Description: The **ResultsHandler** is responsible for aggregating the results returned by one or more search services, represented by their respective **ServiceHandler**.

ServiceHandler
- Type: Class
- Description: The **ServiceHandler** is responsible for receiving the results returned from a particular search service, and converting them into a uniform **ResultSet** representation for further processing within the **ResultsProcessor**. It is an implementation of the **Adapter/Wrapper Design Pattern** [12].

ResultsProcessor
- Type: Class
- Description: The **ResultsProcessor** is responsible for the processing of results that have been passed from the **ResultsHandler**. It is analogous to the **QueryProcessor** in creating a pipeline of **ResultFilters** that allow a variety of post-processing steps to take place prior to return results to the user via the **Client**. As such, The combination of **ResultsProcessor** and **ResultFilter** are an implementation of the **Chain of Responsibility Design Pattern** [12].

ResultsFilter
- Type: Class
- Description: The **ResultsFilter** is responsible for the processing of a set of results, either in isolation or as part of a larger processing pipeline. It closely follows the design of the QueryFilter as described previously. The components can be developed as a local functional units that process the result set, or passed to a remote service that will return a processed set. Although a small part of the overall design, along with the query filters, they combine to facilitate the development and integration of the children-specific information access components developed throughout the project.

ResultsLogger
- Type: Class
- Description: The **ResultsLogger** is responsible for logging result sets associated with queries submitted by users in a datastore for retrieval and analysis.

ResultsDispatcher
- Type: Class
- Description: The **ResultsDispatcher** is responsible for sending the post-processed results to the **Client** for presentation to the user.

5.5 User Component

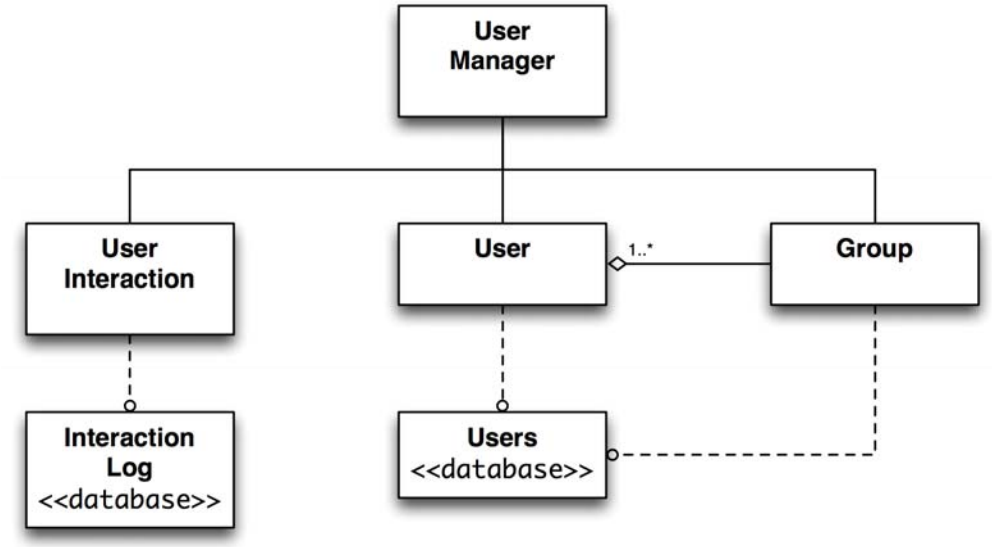

Figure 7: Class diagram for User Component

Design Elements

UserManager

- Type: Class
- Description: The **UserManager** acts as a simplified interface to the rest of the functionality of the **User** component, exposing the major public interfaces, whilst hiding the implementation of the classes. It is an example of the **Façade Design Pattern** [12].

User

- Type: Class
- Description: The **User** is responsible for modelling information about registered users.

Group

- Type: Class
- Description: The **Group** is responsible for modelling collections of users that may be part of a user group using collaborative service.

UserInteraction

- Type: Class
- Description: The **UserInteraction** is responsible for receiving user interaction activity events sent from **ClientInteraction** and logging them into the **InteractionLog**.

6 Information Viewpoint

Viewpoint Description

The purpose of the Information Viewpoint is to describe the model of persistent data used within the framework. Classes within the framework may have to access and update data to achieve their desired functionality. There are six major data entities in the framework that are modelled here: the **user** and associated their **profile**, **groups**, **interactions**, **queries**, **resultsets** and **results**. Figure 8 shows the Entity-Relation Diagram for the data model used by the framework.

Design Concern

Identifies the persistent data used within the framework for management and for logging.

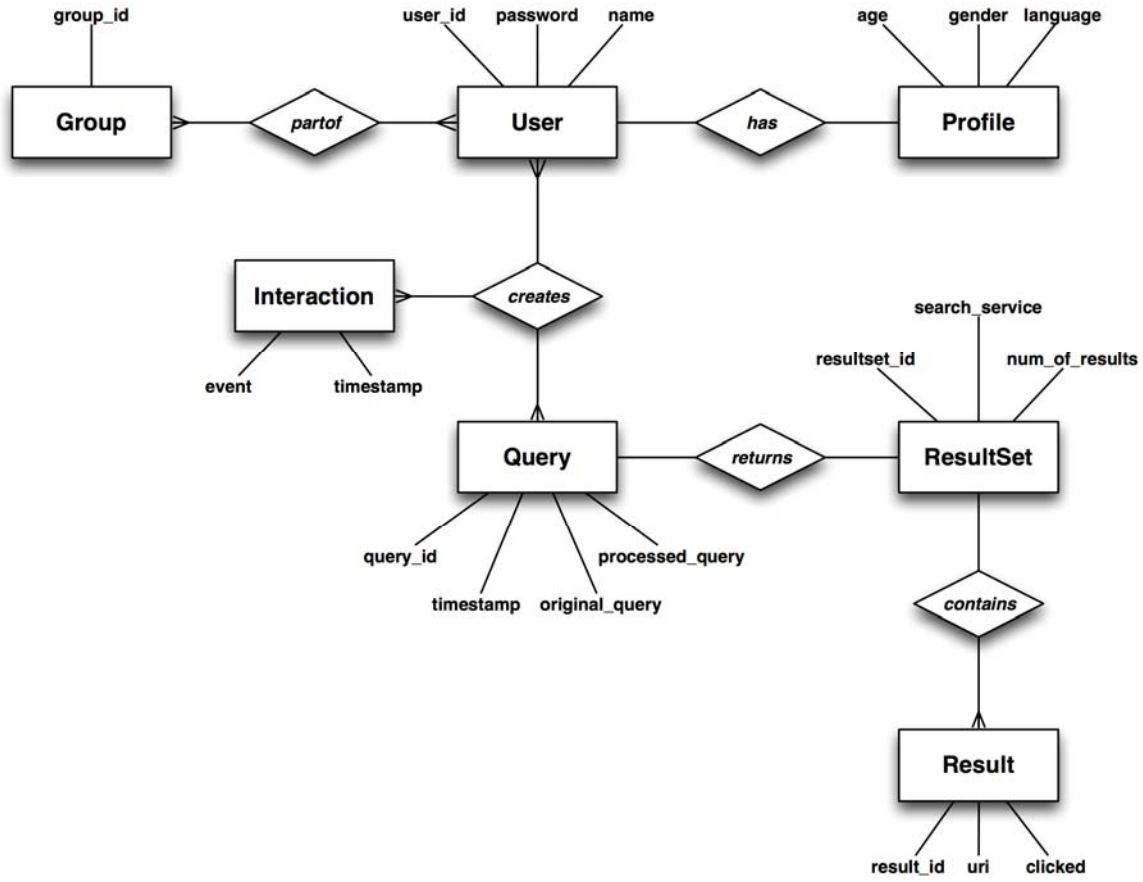

Figure 8: Entity-relation model for PuppyIR service

Design Elements

User

- Type: Data Entity
- Description: A basic representation of a user in the framework. The **user_id** and **password** attributes exist for identification and authentication, whilst the **name** allows for a small amount of personalisation (or set to 'anon' if necessary).

- Attributes:
    - user_id (string)
    - password (string)
    - name (string)
- Relations:
    - each **User** has an associated **Profile** data entity
    - each **User** creates one or more **Query** data entities
    - each **User** can be part of one or more **Groups**
    - each **User** creates **Interaction** events

Profile

- Type: Data Entity
- Description: Whilst not every service will need detailed user profiles, the framework contains a 'basic' profile that can be extended based on a particular service's needs.
- Attributes:
    - age (int)
    - gender (int)
    - language (string)

Group

- Type: DataEntity
- Description: Collections of users may be created and used as part of collaborative services.
    - group_id (int)

Interaction

- Type: DataEntity
- Description: The behaviour of users when using a service via a client is useful information to log and analyse. The framework includes the ability to log the time and type of activity that is taking place (dependent upon what the client choses to consider 'interaction events'.
    - timestamp (datetime)
    - event (string)

Query

- Type: Data Entity
- Description: A representation of a user query for logging purposes. As illustrated in previous a core contribution of the framework to flexibility is the pluggable query-pre-processing. In light of this it is important to preserve the query provenance by maintaining the original query and its processed counterpart.
- Attributes:
    - query_id (int)
    - timestamp (datetime)
    - original_query (string)
    - processed_query (string)
- Relations: each **Query** returns an associated **ResultSet** data entity

ResultSet

- Type: Data Entity
- Description: A representation of the results that were returned for a particular query.
- Attributes:
    - resultset_id (int)
    - search_service (string)
    - num_of_results (int)
- Relation: each **ResultSet** contains zero or more **Result** data entities

Result

- Type: Data Entity
- Description: A representation of a single result from a result set.
- Attributes:
    - result_id (int)
    - uri (string – Uniform Resource Identifier)
    - clicked (Boolean)

Implementation Technologies

WAFs generally include functionality to handle ORM [18] and promote a 'convention over configuration' approach which greatly simplifies and reduces development effort when maintaining a link between the active objects the system requires and their persisted equivalents stored in a relational database.

Logging happens across several components and is a good example of Aspect-Oriented Programming. Rather than re-implement independent Logging tools per component, reuse of the popular **Log4J** library [25] is encouraged to record queries, results and user interaction, and any other aspects of the search process There are two methods of log storage – log to text file or to a relational database (latter is the preferred option for access and scalability). In the latter case, an appropriate open source relational database technology will be used, such as **SQLite** [26] or **MySQL** [27].

7 Interface Viewpoint

Viewpoint Description

The Interface Viewpoint provides components with the means to use each other without necessarily having to be aware of the implementation details, leading to a more loosely coupled system. Figure 9 shows the high-level interface relationships between the components of the framework using a UML Component Diagram.

Design Concern

Identifies the interfaces that the components of the framework expose or require to achieve their functionality.

Design Elements

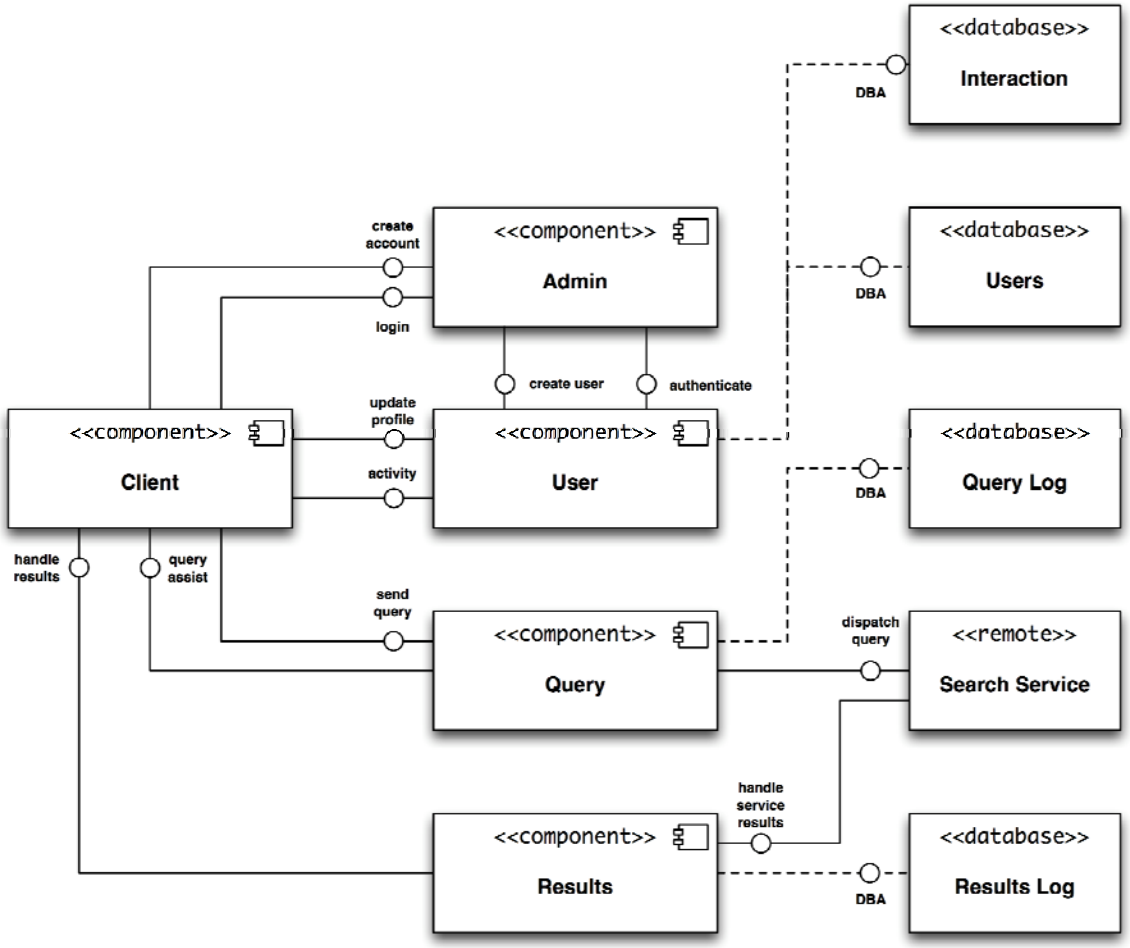

Figure 9: Component Interface Dependencies

Admin

- Exposes:
  - **Create Account** – permit a user to create a new user account with the service, providing a unique *username* and good *password* are supplied.
  - **Login** – permit a user with the correct *username* and *password* pair to log into a service.
  - **Service Configuration** (connectors not shown in Fig. 9) – permit all components to access global service properties: *query and results pipeline configuration.*
- Requires:
  - **Create User (Users)**
  - **Authenticate (Users)**

Client

- Exposes:
  - **Query Assist** – during the process of submitting a query, query suggestions may be sent to the client-side.
  - **Handle Results** – after a query has been submitted, processed, dispatched to a service and the return results processed, a final result set (in Open Search format [15]) is handled on the client-side.
- Requires:
  - **Create Account (Admin)**
  - **Login (Admin)**
  - **Update Profile (User)**
  - **Activity (User)**
  - **Send Query (Query)**

Query

- Exposes:
  - Send Query – accepts a *query* from the client-side, potentially using query assistance or query processing.
- Requires:
  - Search (**Search Service**)
  - Query Assist (**Client**)
  - DBA (**Query Log**) – database access to create, read, update or delete query records when logging search queries.

Results

- Exposes:
  - **Handle Service Results** – handles search results from a specific search service and in a specific format (e.g. Open Search or other result format)
- Requires:
  - **Handle Results (Client)**
  - **DBA (Results Log)** – database access to create, read, update or delete results log records when logging result sets.

User

- Exposes:
  - **Create User** – create a new user account with given information: *username*, *password*, and optional profile information (*age*, *gender*, *language*).
  - **Authenticate** – create a new session for a user with appropriate credentials: *username* and *password*.
  - **Update Profile** – update (or extend) an authenticated user's profile information: *profile key* and *new profile value.*
  - **Activity** – capture interaction events sent from user interface and store in InteractionLog
- Requires:
  - **DBA (Users)** – database access to create, read, update or delete user records when creating user accounts, authenticating user credentials and updating profile information.

8 Interaction Viewpoint

Viewpoint Description

The Interaction Viewpoint defines strategies, or sequences of control, amongst the entities within the overall framework when carrying out their required functionality.

Design Concern

Identifying the flow of information, control and interaction between entities throughout the framework

Design Elements

The following sections describe the high-level interaction for User Search Interaction (Sec. 8.1), Query Interaction (Sec. 8.2), and Results Interaction (Sec. 8.3). Each Interaction is shown using a UML Sequence Diagram (Fig. 10 – 12).

8.1 User Search Interaction

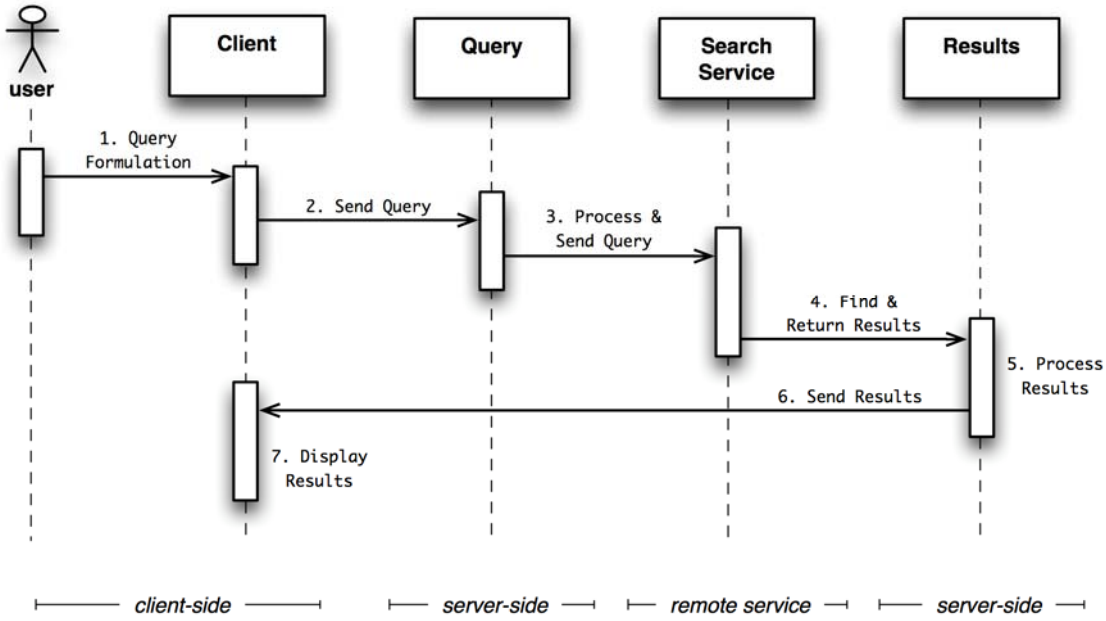

Figure 10: Sequence Diagram for User Search Interaction

The sequence diagram[1] shown in Fig. 10 illustrates the high-level interaction between the user, components of the framework (Client, Query, Results), and external resources (a search service) when a user search interaction occurs. Rather than specify the exact low-level method calls, each interaction link indicates the overall behaviour of what is occurring at each stage. The forthcoming sequence diagrams will break these stages down into more granular detail.

1. **Query Formulation**: The interaction begins by a user formulating their query via the user interface that the Client presents.

[1] The labelled bars at the bottom of the diagram are an extension to the standard UML 2 Sequence Diagram, included here to conveniently indicate the relationship between the components and their expected physical deployment.

2. **Send Query**: Once the query has been completed and submitted by the user, it is transmitted to and handled by the Query component of the framework.

3. **Process and Send Query**: At this stage the Query component can potentially pre-process the query before sending it onto a designated search service. It also logs the query within a local datastore for later analysis. Once processing and logging has completed, the query is dispatched to the search service, which is potentially remotely located.

4. **Find and Return Results**: The search service is outside the control of the framework, and the query is sent and handlers are set up to respond to the presence or absence of results returned by a search service.

5. **Process Results**: The Results component handles the result set returned by a search service. It can potentially post-process the set of results depending upon the configuration that has been setup. The result set is also logged at this stage for later analysis. After logging and processing, the final result set is converted to an open format for search engine results and sent to the client for presentation.

6. **Send Results**: The Results component sends results back to the Client component.

7. **Display Results**: The Client component presents the results that have been sent for the user to interact with.

8.2 Query Interaction

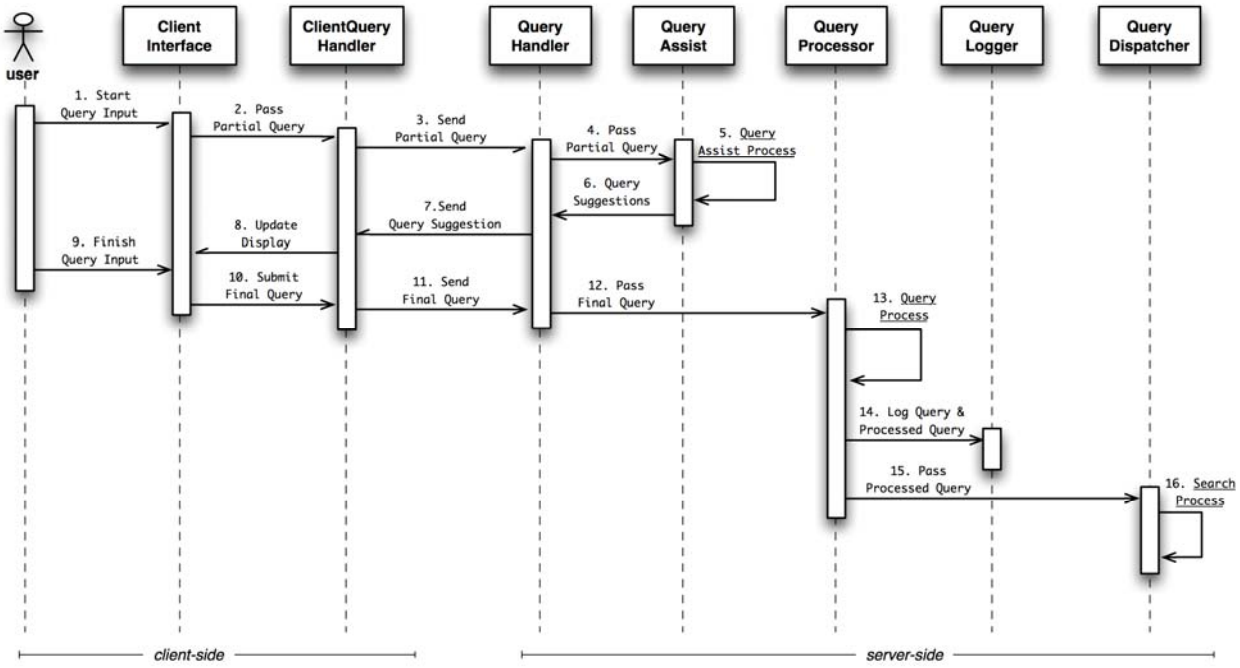

Figure 11: Sequence diagram for Query Interaction

The sequence diagram shown in Fig. 11 illustrates the series of interactions that take place between the classes and sub-components of various components within the framework when dealing with a query from the user. For clarity, interactions are directly linked between relevant classes, however, in some cases there may be a mediator class that coordinates the interactions, especially between classes that are found in different components. As mentioned in the previous

section, whilst more detailed, the interactions here need to be mapped to actual method calls. In many cases the importance here is to illustrate the information that is being passed throughout the interaction

1. **Start Query Input**: A user begins to type (or select in the case of pre-written text query or image). This creates an event within the ClientInterface.

2. **Pass Partial Query**: The ClientQuery class is an observer of the ClientInterface and reacts by beginning an asynchronous process in which the partial or selected query recorded so far is sent to the server-side to activate query assistance.

3. **Send Partial Query**: The partial query is sent from the ClientQueryHandler to the QueryHandler on the server-side.

4. **Pass Partial Query**: The partial query is passed to the QueryAssist class.

5. **Query Assist Process**: Depending upon how the developer has implemented their query assistance process, a list of query suggestions will be created.

6. **Query Suggestions**: The QuerySuggestions are passed to the QueryHandler.

7. **Send Query Suggestions**: The QueryHandler sends the QuerySuggestions back to the ClientQueryHandler.

8. **Update Display**: The ClientQueryHandler causes the ClientInterface to update its view in light of the QuerySuggestions received.

9. **Finish Query Input**: The user, potentially using the QuerySuggestions, or continuing with their own query formulation process, completes their query.

10. **Submit Final Query**: The ClientInterface submits the final query when the user indicates they are done (e.g. by typing return or clicking a ‘search’ widget in the user interface).

11. **Send Final Query**: The ClientQueryHandler sends the query to the QueryHandler on the server-side.

12. **Pass Final Query**: The QueryHandler passes the query to the QueryProcessor.

13. **Query Process**: The QueryProcessor can subject the query to a processing pipeline to alter its contents before submitting it to a search service. Although not shown here, there could be zero or more QueryFilters that have been developed independently and integrated into this search service.

14. **Log Query & Processed Query**: The QueryLogger logs both the original query submitted by the user and the processed query for later analysis.

15. **Pass Processed Query**: The processed query is passed to the QueryDispatcher.

16. **Search Process**: The QueryDispatcher sends the query to the appropriate SearchService as indicated by the ServiceConfiguration.

8.3 Results Interaction

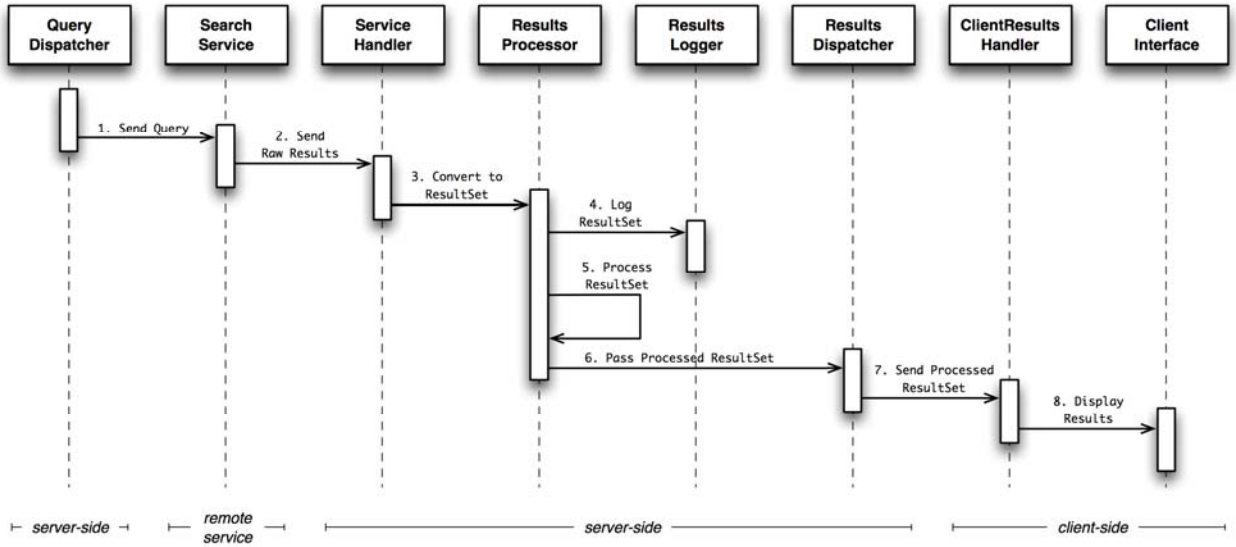

Figure 12: Sequence diagram for Results Interaction

The sequence diagram shown in Fig. 12 illustrates the series of interactions that take place between the classes and sub-components of the components when returning a set of results to the user after a query has been submitted to a search service. As before there is a mixture of information passing and interaction within the diagram.

1. **Send Query**: The QueryDispatcher sends the final user query to the designated search service.

2. **Send Raw Results**: The SearchService sends its results to a corresponding ServiceHandler that was associated with a particular query interaction (i.e. the QueryDispatcher creates the appropriate handler when it dispatches a query).

3. **Convert to ResultSet**: Each ServiceHandler for a particular search service can convert the results that are returned to the ResultSet datastructure that is used internally with the framework, and allows multiple ResultFilters to operate on any results from any search service with a corresponding ServiceHandler.

4. **Log Result Set**: The original ResultSet is logged.

5. **Process Result Set**: The ResultsProcessor can subject the original ResultSet to a pipline of processing operations as specified by the ServiceConfiguration. Although not shown here, there could be zero or more ResultFilters that have been developed independently and integrated to this search service.

6. **Pass Processed Result Set**: The processed ResultSet is passed from the ResultsProcessor to the ResultsDispatcher once all post-processing operations have completed.

7. **Send Processed Result Set**: The ResultsDispatcher sends the processed ResultSet from the server-side to the ClientResultsHandler on the client-side.

8. **Display Results**: The ClientResultsHandler updates its model of the ResultSet causing the ClientInterface to updates its view of these results and present them to the user.

9 Design Rationale

As stated in Sec. 2, efforts have been made to avoid ever-engineering the design of the framework by detailing every last specific detail. Instead, the aim has been to describe the important concerns of the framework design, leaving flexibility to incorporate alterations that become necessary through the experience of developing the core framework. Major revisions to design will be reported in future deliverables on the progress of implementation.

The key focus instead has been to identify the main regions of flexibility in the framework, and describe how these hot-spots integrate into a larger piece of software. These important areas are:

- supporting multiple clients
- supporting alternative methods of query assistance
- creating pipelines of query processing filters
- creating pipelines of result processing filters
- providing wrappers for multiple search services

By exposing these regions of flexibility, the framework should provide a solid platform for 3rd party developers to build novel and assistive information access services for children, without having to start from first principles. As many areas have been purposefully designed to support loose coupling through the use of appropriate design principles and patterns, it is possible to envision an eventual library of components that can be plugged in to many services, further reducing the development effort.

10 Implementation Plan

The following is a suggested outline plan for implementation:

1. Make fundamental implementation choices (e.g. programming language, web application framework, integration frameworks, and other useful toolkits) (**WP4 – Task 4B.1**)

2. Create wrappers for search services, publish interface for other project members to use (**WP4 – Task 4A.2**)

3. Build Query Component – publish interfaces for Query Assistance and Query Filters (**WP4 – Task 4B.4**)

4. Build Results Component – publish interfaces for Result Filters (**WP4 – Task 4B.4**)

5. Build Client Component – release components for building user interfaces (**WP4 – Task 4B.2 & Task 4B.3**)

6. Create support for multiple users & groups (User component), administration and service configuration (Admin Component)

11 Summary

This report has detailed the software design description for the PuppyIR Framework. The design was mainly derived from **D1.2 – User Scenario and Requirements, D1.3 – Technical Requirements** and **D4.1 – Specification Report**. The purpose has been to describe in detail the design and implementation options for the development of the framework.

The design approach adopted was standards orientated, yet flexible enough to permit easy communication of ideas whilst allowing for evolution of design based upon the experience that will be gained from implementation. The proviso throughout has been to avoid over-engineering and highlight the key *regions of flexibility* of the framework that allow developers to make multiple distinct services from the framework.

The major elements of the design were communicated using a sequence of standard design viewpoints appropriate for the project (Composition, Logical, Information, Interface and Interaction). For each design viewpoint, suitable design languages such as UML and ERD were used to communicate required components to achieve the design. Where appropriate, suggestions were made as to the types of technologies and code re-use that would allow the development of the framework to be accelerated. Finally a brief outline of the implementation plan was presented.

The contents of this report will provide the foundation for the forthcoming deliverable, **D4.3 – Report on Implementation and Documentation**, which will report upon the implementation phase of the project, and provide any updates to the system design description based upon the experience gathered.

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] **IEEE Standard for Information Technology – Systems Design – Software Design Descriptions / IEEE Std 1016-2009,** DOI: 10.1109/IEEESTD.2009.5167255

[7] **UML Distilled: Brief Guide to the Standard Object Modeling Language**, M. Fowler, 2003

[8] **The Entity Relationship Model - Towards a Unified View of Data**. P. P. Chen, ACM Transactions on Database Systems, 1(1):p9–36, March 1976

[9] **UML Software Architecture and Description**, C. Lange, M. Chaudron and J. Muskens, IEEE Software, 23(02):p40-46, 2006

[10] **Software Architecture: Foundations, Theory and Practice**, R. Taylor, N. Medvidovic, and E. Dashofy, John Wily and Sons, 2010

[11] **Architectural Manifesto - Designing Software Architectures**, M. Kontio, IBM Dev, 2004

[12] **Design Patterns: Elements of Reusable Object-Oriented Software**, E. Gamma, R. Helm, R. Johnson, and J. Vlissides, 1995

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

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

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

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

[17] **Principled Design of the Modern Web Architecture (REST)**, R. Fielding, R. Taylor and N. Richard, ACM Transactions on Internet Technology, 2(2):115–150, 2002

[18] **Hibernate** (Object-Relation Mapping & Persistence Framework): http://www.hibernate.org/

[19] Inversion of Control / Dependency Injection (Design Principle): http://martinfowler.com/articles/injection.html#InversionOfControl

[20] **Groovy/Grails** (Web Application Framework): http://www.grails.org/

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

[22] **Ruby on Rails** (Web Application Framework): http://rubyonrails.org/

[23] **Scala/Lift** (Web Application Framework): http://liftweb.net/

[24] **Spring** (Application Integration and Configuration): http://www.springsource.org/

[25] **Log4J** (Logging Framework): http://logging.apache.org/log4j/1.2/

[26] **SQLite** (Lightweight Relational Database): www.sqlite.org/

[27] **MySQL** (Fully Featured RDBMS): www.mysql.com