The CASPAR Finding Aids

Henri Avancini, Carlo Meghini, Loredana Versienti

CNR-ISTI

Area dell Ricerca di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy

EMail: Full.Name@isti.cnr.it

ABSTRACT

CASPAR is a EU co-funded Integrated Project aiming at developing a methodology and a set of tools, the CASPAR key components, for the lomg-term preservation of digital information. The methodology is based on the OAIS Reference Model and will be validated, along with the tools produced by CASPAR, by application on test-beds coming from three different domains: the cultural, artistic and scientific domains.

One of the CASPAR key components is the FIND Components, which is implementing the Finding Aids of OAIS. According to OAIS, FIND manages Description Information that is associated to Information Packages in order to support the discovery of those Packages. FIND allows defining, creating, persisting, querying, browsing, and accessing both Description Information and the association between Information Packages and Description Information.

This paper illustrates the requirements that have been considered in the design of FIND; provides a detailed description of such design, highlighting its underlying conceptual model and the resulting API; then it gives some details about the implementation of that API; finally it shows how the FIND has been employed in realizing significant preservation scenarios within the CASPAR test-beds.

Overview

The OAIS Reference Model [1] is an ISO Standard providing a framework for the understanding of the concepts needed for the long-term digital information preservation and access. The Reference Model includes, among other things, a functional model of the archive, identifying six main functional areas: Ingest, Data Management, Archival Storage, Access, Preservation Planning and Administration. The Model describes each area in detail, outlining the main services provided by the area in terms of a set of interrelated functions. Data Management, in particular, is concerned with "the services and functions for populating, maintaining, and accessing Descriptive Information which identifies and documents archive holdings" . According to the OAIS Information Model, Descriptive Information is the set of information, which "is provided to Data Management to support the finding, ordering, and retrieving of OAIS information holdings by Consumers".

The CASPAR Integrated Project (http://www.casparpreserves.eu/) purports at creating a set of key components implementing crucial functions of the OAIS Reference Model. This paper describes FIND, the CASPAR key component that implements the Data Management functionality of the OAIS model.

The basic challenge faced by FIND is to achieve various forms of independence, in order to be usable in the widest possible range of archives, included those which would like to enhance the finding aids that are already in place. The two basic kinds of independence aimed at by FIND is the independence from the Data Definition Language and from the Data Manipulation Language, the latter including the Query Language. A further requirement that has been considered, is the richness of expressivity of the language for representing Description Information; also this requirement is dictated by generality, i.e. to be able to accommodate in FIND the widest possible range of archives, some of which, notably those in the artistic domain, require a high expressive power. Finally, adherence to standards has been considered, in order to favour wide adoption and a long lifetime to FIND.

The resulting architecture includes two component types: the Findind Manager, directly responsible for managing Description Information, and the Finding Registry, for managing Finding Managers. Every Finding Manager accepts Description Information in a specific Data Language and supports a specific Query Language. Different Finding Managers may support different languages, and this implies language independence. CASPAR has implemented an RDF-based Finding Manager [2], thus fulfilling also the expressivity and the adherence to standard requirments. Both the Finding Manager and Registry expose their API as web services.

The paper is organized as follows. Next Section gives the conceptual model of FIND, describing the concepts that have been developed in order to fulfill the above requirements. The architecture of the FIND component is subsequently introduced, for a specific choice of the Data Definition and the Data Manipulation Language. An overview of the usage of FIND within the CASPAR project is finally presented.

The conceptual model

The CASPAR Finding Aids is a software package that provides the functionality for the discovery of Archival Information Packages (AIPs) according to the OAIS Reference Model.

In order to achieve its goal, the FA is based on two basic components:

1. Finding Registry, and
2. Finding Manager.

A *Finding Registry* supports the publication and discovery of Finding Managers, in the same way a UDDI server supports the publication and discovery of Web Services. In CASPAR, the concept of Finding Registry is introduced in order to be independent from any specific technology or *de facto* standard.

From a functional point of view, a Finding Registry supports two main functionalities:

1. Management of Finding Managers by Registration, Deregistration, Discovery and Browse of Finding Managers.
2. Indexing and retrieval of all the Description Information objects owned by the Finding Managers registered with the Finding Registry.

A Finding Manager registers by providing a description of itself to the Registry. This description (FMDesc) contains required information, such as:

- (Data definition & query) language spoken by the Finding Manager.
- Handle for invoking the Finding Manager.

Additionally, an FMDesc object can contain information concerning properties of the Finding Manager that applications consider useful for discovery purposes.

A *Finding Manager* supports the management of Description Information (DescInfo, for short), and is bound to a language for defining and for querying DescInfo. For example, a Finding Manager may talk SQL, another one RDF/S [3], and another one XML. Every Finding Manager registers with at least a Finding Registry in order to be discovered by applications.

From a functional point of view, a Finding Manager supports three main functionalities by using a Data Manipulation Language:

1. Management of DescInfo:
    a. At the schema level: create, delete and browse DescInfo schema elements (i.e., tables or classes or DDTs).
    b. At the object level: create, delete, update and browse DescInfo objects (*i.e.*, tuples or objects or documents).
2. Management of the association between DescInfo objects and AIP identifiers, including usage of these associations for AIP discovery:
    a. Create, delete query and browse association instances (*i.e.* (AIP-id, DescInfo-id) pairs).
    b. Discovery of AIPs via queries on DescInfo objects.
3. Primitives for the preservation of DescInfo, at all levels. These are: export and import of all of the above: DescInfo schema elements, objects and associations with AIPs.

The following UML class diagram (Figure 1) illustrates the relations amongst the entities introduced so far within a CASPAR installation.

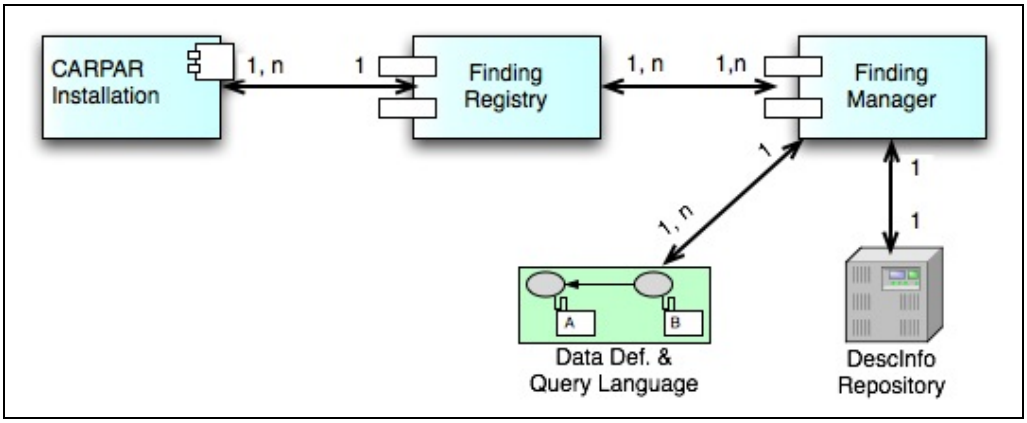

Figure 1 - Finding Registries and Finding Managers

A user application interacts with the Finding Manager using a particular Query Language (QL) for querying and/or browsing the DescInfo Repository.

Every CASPAR installation has at least a Finding Registry, while a Finding Registry can only serve a single installation. A Finding Registry registers at least one Finding Manager, while a Finding

Manager may be registered with more than one Registry for reliability reasons. Every Finding Manager has exactly one Language, while several managers can talk the same language. Finally, every Manager manages exactly one DescInfo Repository (set of schema elements and associated objects), and vice versa a DescInfo Repository belongs to exactly one Manager.

Below, Figure 2 sketch relations between above-mentioned Data Definition and Query languages, Descriptive Information schema and object, and Archival Information Packages.

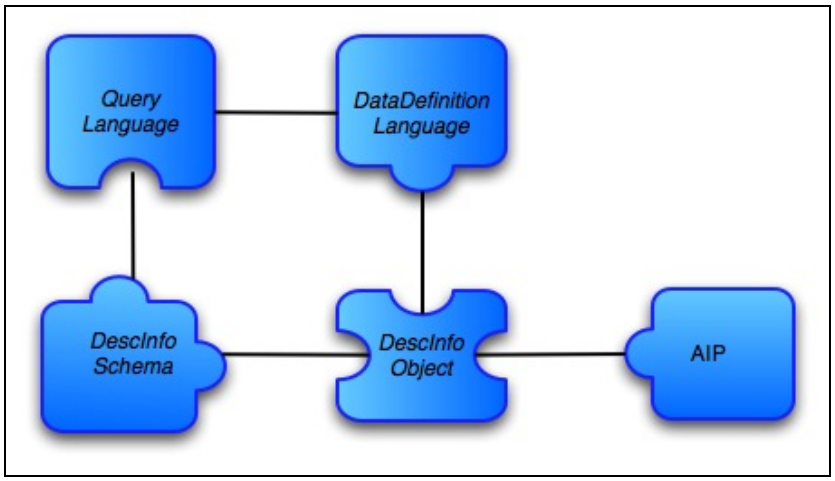

Figure 2 – Finding Aids conceptual components model

Finally, on Figure 3 the Finding Aids overall interface is presented thought the Finding Manager and Finding Registry components.

Finding Manager

Four main abstract classes form Finding Manager logical component (Figure 4):

- FindingManager abstract class is responsible for FM configuration data and for parse queries. It is also in charge of providing access to the Result Set.
- DescInfoSchemaManager abstract class provides the functionality to create, delete, and list DescInfo stored schemas.
- DescInfoObjectManager abstract class manages the DescInfo object itself, which includes the association with the corresponding AIP.
- DescInfoRepository abstract class is responsible for the FM storage. Three main concepts are stored: (a) DescInfo schemas, (b) DescInfo objects, (c) specific FM information, including AIP objects and their association with DescInfo Objects.

To instantiate a concrete Finding Manager designer should implement it by extending described abstract classes. To do so a concrete Data Definition Language and a concrete Query language need to be chosen. By fixing them the Finding Manager can manage different types of DescInfo objects, by storing one schema element for each type.

We instantiated a concrete Finding Manager by using, at the Data Layer, the SWKM [4] (see Section Architecture below for details), using RDF as Data Definition Language and RQL as Query Language (Figure 5). Light blue classes are the provided implementation, and the rest of the classes are the FR foundation classes.

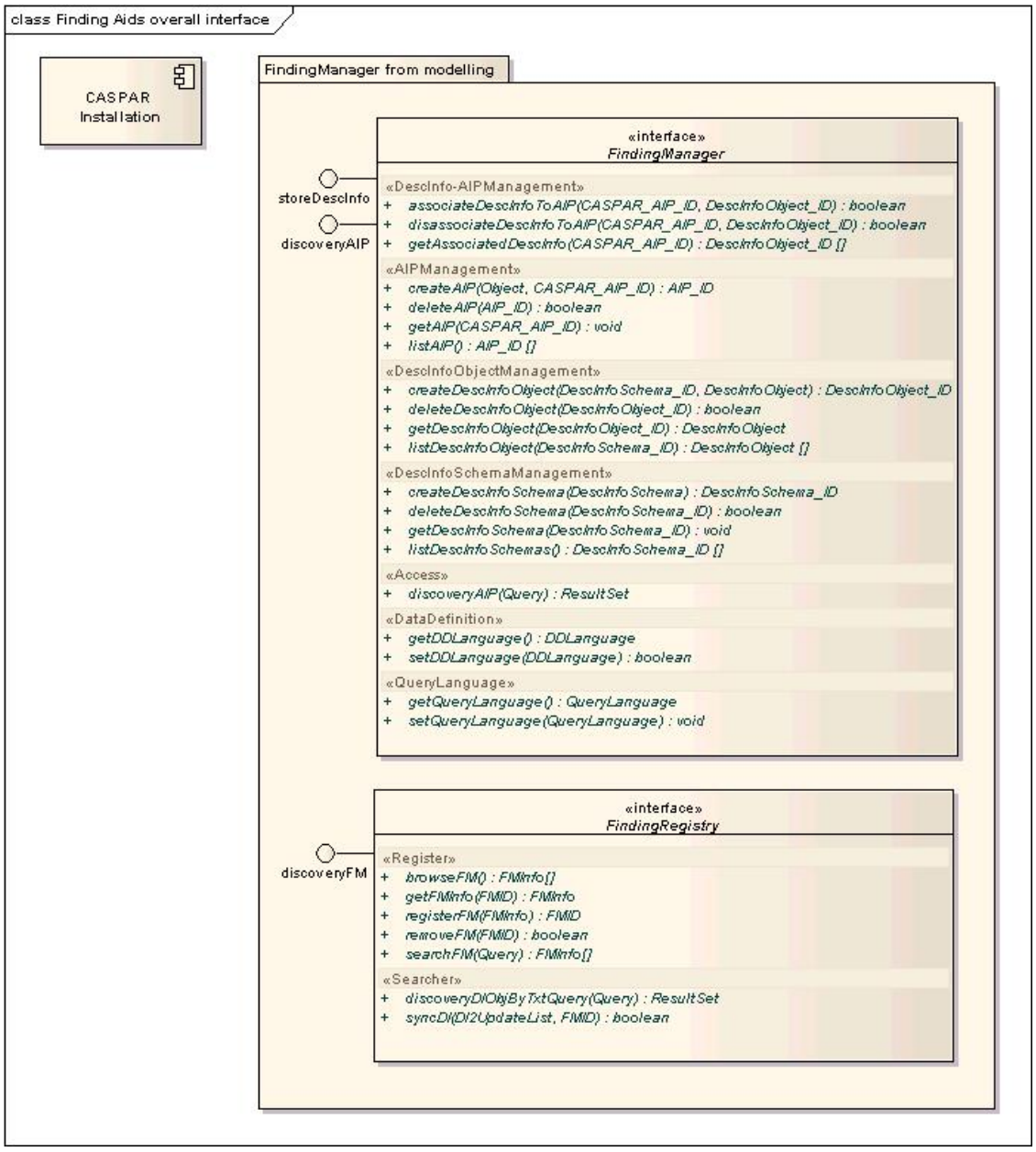

Figure 3 - Finding Aids overall interface

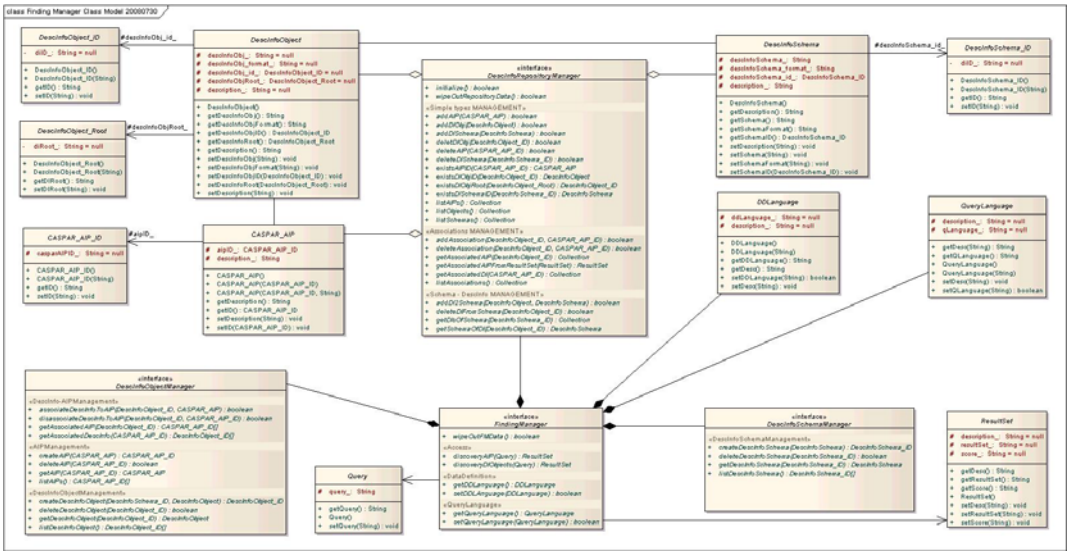

Figure 4 – Finding Manager Logical View

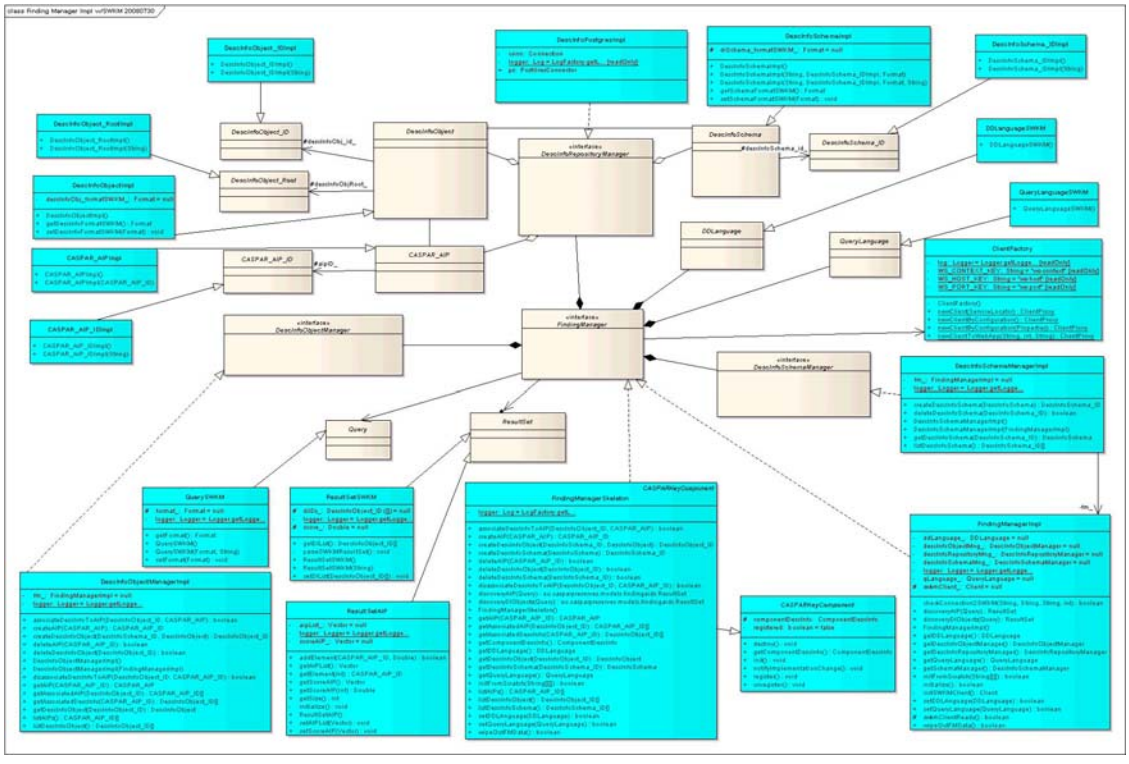

Figure 5 – Finding Manager with RDF and RQL

Finding Registry

Finding Registry has two main functionalities areas. First, the Finding Managers management (RegisterFM class); and second, the DescInfo indexing support (DIIndexer class). Complete the main Finding Registry design the extension from CASPARKeyComponent class, as the Finding Registry is a CASPAR component.

Finding Manager management functionalities provides basic functionality to register FMs and to query for registered Managers.

- Save FM, allows registering a new deployed Finding Manager into the Finding Registry. For Web service based Finding Managers following information need to be provided: Data Definition Language (ie. RDF/S), Query Language (ie. RQL), server URI and WSDL. An ID is assigned to new registered Finding Manager.
- Delete FM, is used to cancel a registered FM from the Finding Registry by a given FM ID.
- Browse FMs, this functionality shows all registered Finding Managers with the corresponding data, ie. assigned ID, Data Definition Language, Query Language, server host, and WSDL.
- Search FMs, provides the functionality to query the Finding Registry by providing Data Definition Language and/or Query Language. A list of Finding Managers that satisfy query is returned.
- Get FM, allows user to get Finding Manager registration information (Data Definition Language, Query Language, server host, and WSDL) from a valid Finding Manager ID.

DescInfo indexer functionalities area allows user to directly query the Registry for a particular DescInfo thought a simple text query. Moreover this functionality supports Finding Managers on indexing DescInfo to support text queries at FM level.

- Search DescInfo, by performing a full text query the system return the list of DescInfo that satisfy the query by providing added information also, like Finding Managers that contains that DescInfo objects.

On Figure 6 we show the implementation we made. Light blue classes are the provided implementation, and the rest of the classes are the FR foundation classes. This implementation use SOLR as textual indexer server (see Section Architecture for details).

All FIND functionality (Finding Registry and Finding Manager) can be accessed both by a client-application doing Web Service calls or by the GUI ‘CASPAR Web Desktop’.

Architecture

The system of archiving FIND requirements is by using stable technologies, as Web Services, RDF, etc. but this do not limit the scope of the overall Finding Aids architecture which is to stands the base for a long term preservation of DescInfo and related AIPs. This Section present the characteristics of the Finding Aids that is being implemented within the CASPAR project to support the discovery of AIPs according to the OAIS Reference Model and the conceptual model and interfaces presented so far.

Figure 7 shows the technological approach employed on the Finding Aids that involves three different levels: (a) Data layer, (b) Business layer, and (c) User Interface layer.

At a certain time a Finding Aids CASPAR component is deployed and configured to interact with the rest of a CASPAR installation, i.e. the minimum set of CASPAR components needed for digital preservation.

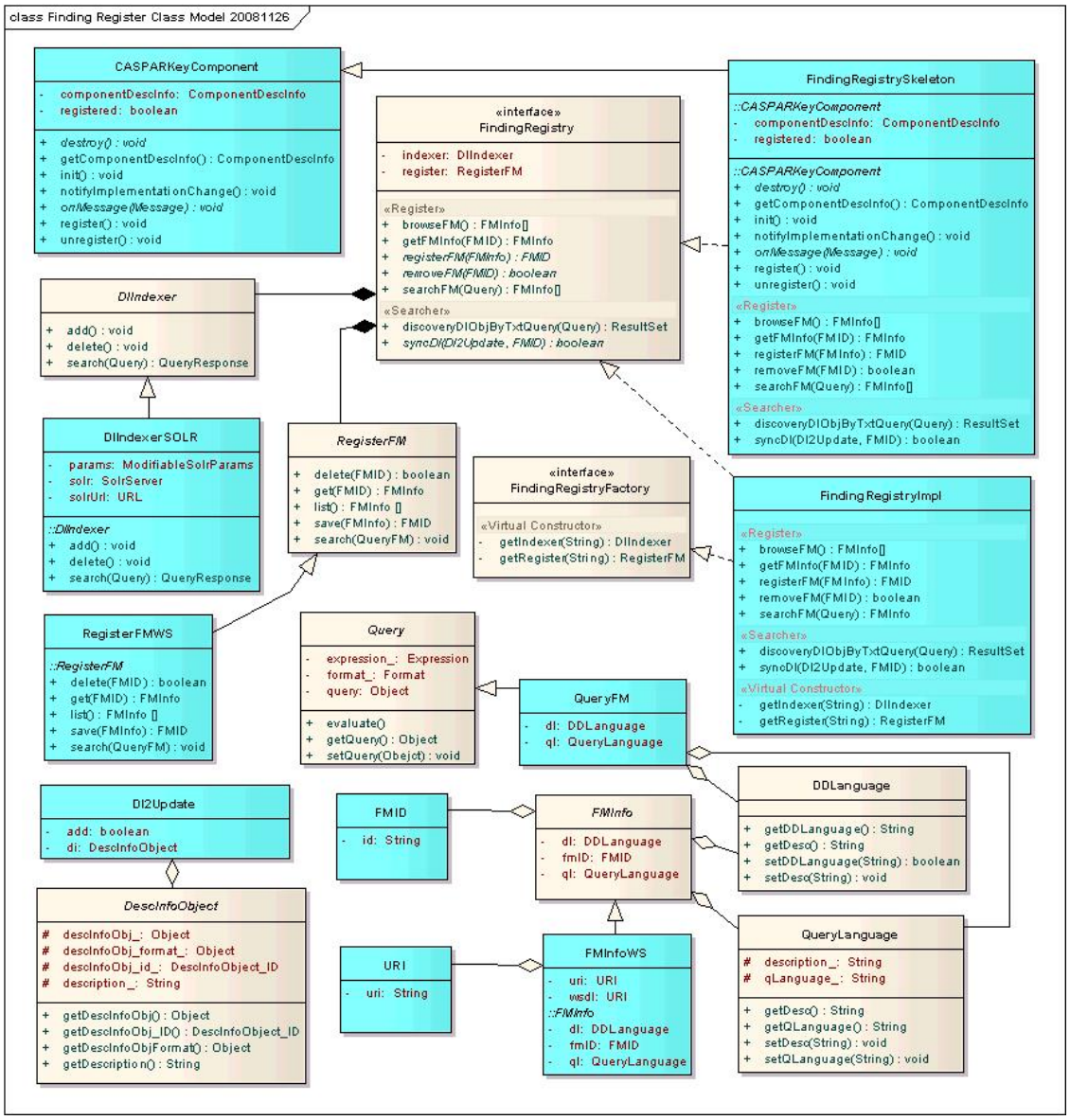

Figure 6 – Finding Registry with SOLR

An user, the Producer, provide digital objects to the CASPAR infrastructure by creating an Archival Information Packages (AIP) and storing it on CASPAR. By interacting with Finding Manager UI layer, the Producer user associate specific Descriptive Information (DescInfo) to created AIPs. In fact, Finding Aids logic allows storing DescInfo objects, the related AIP, and other Data relevant for preservation purposes (Business and Data layers).

The Finding Aids supports the Consumer user in locating the relevant data. The Consumer user (application) is responsible to provide access to Finding Aids overall interface in order to allow to search AIPs based on conditions upon the Descriptive Information, s/he specifies a certain Query and the Finding Aids answer with the corresponding Result Set based on stored DescInfo information and associated AIPs.

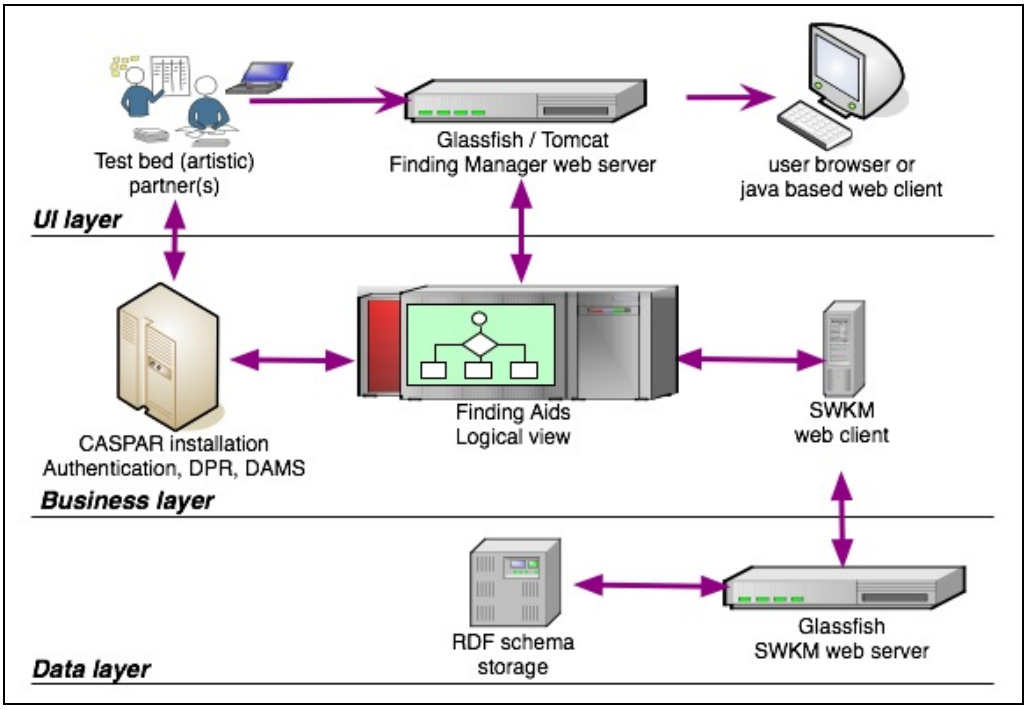

Figure 7 – Finding Aids technological diagram (example with SWKM)

One of the implementations this architecture uses both:

- the Semantic Web Knowledge Middleware (SWKM) on Finding Manager to preserve internal data structures as well as DescInfo object structure (graphspaces) and documents (namespaces), and the
- search server SOLR for DescInfo textual indexing on Finding Registry.

Implemented Finding Manager has RDF as Data Language and accept RQL queries, it uses the SWKM to store Descriptive Information objects and to query them.

Semantic Web Knowledge Middleware (SWKM) provides a set of core services for managing Semantic Web data, i.e. it is a set of tools for the parsing, storage, manipulation and querying of Semantic Web (RDF) Knowledgebases. In addition, it offers advanced facilites for the comparison of different knowledgebases and the versioning of the schemas stored in it.

Each implemented Finding Manager contains a component that is a SWKM client and it is initialized as follows:

```
DbSettings db = new DbSettings();
db.setDbName( dbName );
db.setUsername( dbUsr );
db.setPassword( dbPsw );
db.setHost( dbHost );
db.setPort( dbPort );
db.setProtocol( dbProtocol );
db.setRepresentation( DbRepresentation.HYBRID);
swkmClient = ClientFactory.newClientToWebApp (
       swkmHost, swkmPort, swkmContext ).with(db);
```

Storing a new DescInfo schema and DescInfo object is performed by using the SWKM client as follows:

- DescInfo schemaID: create a new schema for that object or get the schema ID of a previously stored schema.

```
Collection<RdfDocument> col= new ArrayList<RdfDocument>();
RdfDocument rdfdoc = new RdfDocument(
       (String)descInfoSchema.getSchemaID().getID(),
              (String)descInfoSchema.getSchema(),
              (Format)(descInfoSchema).getSchemaFormatSWKM() );
col.add ( rdfdoc);
fm_.swkmClient_.importer().store(col, Deps.WITHOUT);
```

- DescInfo object: create a new descriptive information object based on information provided as well as AIP object used from FM point of view. Associate these objects.

```
Collection<RdfDocument> col= new ArrayList<RdfDocument>();
RdfDocument rdfdoc = new RdfDocument(
       ((DescInfoObjectImpl)descInfoObj).getDescInfoRoot().getDIRoot(),
              (String)descInfoObj.getDescInfoObj(),
              (Format)(descInfoObj).getDescInfoFormatSWKM() );
col.add ( rdfdoc );
fm_.swkmClient_.importer().store(col, Deps.WITH);
```

Using the SWKM client as follows performs querying DescInfo objects on Finding Managers:

```
ResultSet result = swkmClient_.query().query (
        ((QuerySWKM)query).getFormat(),
        ((QuerySWKM)query).getQuery() );
rsSWKM = new ResultSetSWKM ( result );
rsSWKM.getDIList();
```

DescInfo Indexer implementation is based on SOLR, and is used by Finding Registry to index all DescInfo objects imported on any registered Finding Manager.

SOLR is a standalone enterprise search server with web-services like API. It is based on the Lucene Java search library. DescInfo objects are indexed via XML over HTTP. When Finding Registry receives a DescInfo object from a registered Finding Manager, it parses it creating an XML and adding necessary information to the object like FM ID.

After indexing some DescInfo objects query is possible by calling the Finding Registry web service method or by interaction with the CASPAR Web Desktop. Internally the query is forwarded to the SOLR server via HTTP GET and receives the result as an XML. DIIndexer component format the query in order to have independence between current used indexer (SOLR) and the rest of the FIND component. In this way future change of the Indexer is straightforward because do not affect the rest of the FIND implementation.

DIIndexer supports cursors over formatted result set, so caller can specify the maximum number of elements that get after each call.

This functionality is also used internally by FIND component to support full text search directly from each registered Finding Manager. Calls between sub-components (FM-FR) are transparent to user

and automatic filter by FM ID is performed in order to only return DescInfo objects from calling Finding Manager.

Use within CASPAR

Designated Community Partners (DC) are the real users and data holders of Finding Aids component, they also collaborate also on defining Finding Aids functionality.

From each Partner we collect the used DescInfo schemas and DescoInfo objects. The following table gives the CASPAR communities using the FIND and for each community highlights the schema used for DescInfo, providing links to the corresponding objects.

Community	DescInfo Schemas	Schema links
ESA (Scientific)	Scientific schema (ad-hoc RDF schema)	http://rdfs.esrin.esa.int/EGOC.rdfs#
IRCAM (Artistic)	CIDOC CRM [5] FRBR extension	http://cidoc.ics.forth.gr/rdfs/CIDOC4.3.rdfs# http://cidoc.ics.forth.gr/rdfs/caspar/frbr.rdfs#
UNESCO (Cultural)	CIDOC-CRM Extension for ESRI ASCII Grid data objects CIDOC Extension for UNESCO automatically generated from XML	http://www.casparpreserves.eu/testbed/cultural/esrigrid http://www.casparpreserves.eu/testbed/cultural/ewe/epdl
Univ. of Leeds (Artistic)	CIDOC CRM FRBR extension	http://cidoc.ics.forth.gr/rdfs/CIDOC4.3.rdfs# http://cidoc.ics.forth.gr/rdfs/caspar/frbr.rdfs#

Conclusions e future developments

After succesfull implementation and exploitation of the FIND component in the CASPAR architecture, we plan to have more deployment and to support Finding Managers with different data language than RDF.

References

[1] Reference Model for an Open Archival Information System (OAIS). CCSDS 650.0-B-1, Blue Book, January 2002

[2] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, WWW Consortium, February 2004. http://www.w3.org/TR/rdf-primer/.

[3] Patrick Hayes. RDF Semantics. W3C Recommendation, WWW Consortium, February 2004. http://www.w3.org/TR/rdf-mt/.

[4] The ICS-FORTH RDFSuite: High-level Scalable Tools for the Semantic Web. http://139.91.183.30:9090/RDF/

[5] Martin Doerr. The CIDOC conceptual reference model: An ontological approach to semantic interoperability of metadata. AI Magazine, 24(3):75–92, 2003.