ICADC

Technical Specification Document

Contents

Tables/Figures

1 Introduction

1.1 Purpose of this document

The purpose of this document is to outline a technical solution for handling display of dynamic content on the current implementation of the ICA (ICADC).

In order to cover the required functionality, this document will focus on describing the solution that replaces the phased out FEP application that produced dynamic content for CORDIS.

The term FEP (Filtered Entry Point) is used to describe functionality where the content of the CORDIS databases is displayed by various services of the CORDIS web site. In common usage, this term refers to browseable content and to search functionality implemented using the CGI search technique on DBS.

1.2 Scope of the document

This document explores an alternative architecture for handling the display of dynamic content for CORDIS from a technical point of view. It also presents an analysis of the impact of this architecture on CORDIS as a whole and suggests future migration steps.

1.3 Definitions, Acronyms and Abbreviations

The following table presents the most significant terms used in this document:

Term	Definition
ICA	Integrated CORDIS Architecture.
ICA2	Integrated CORDIS Architecture 2.
ICADC	Integrated CORDIS Architecture - Dynamic Content (application).
FEP	Filtered Entry Point.
CGI	Common Gateway Interface.
DBS	The Fulcrum database server on CORDIS.
CMS	Content Management System.
XML	Extensible Mark-up Language. XML is a W3C recommendation for creating special-purpose mark-up languages. It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of structured text and information across the Internet. Languages based on XML (for example, RDF, RSS, MathML, XSIL and SVG) are themselves described in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form.
XSL	Extensible Style sheet Language. XSL is a set of language technologies for defining XML document transformation and presentation.
XSLT	XSL Transformations. XSLT, is an XML mark-up language used for transforming XML documents. It is the XML transformation language part of the XSL specification (the other parts being XSL-FO and XPath).

Term	Definition
SAX	Serial Access parser for XML or Simple API for XML. SAX is a common interface implemented for many different XML parsers, just as the JDBC is a common interface implemented for many different relational databases.
SAX Event	SAX is an event-driven interface in which an application supplies the parser with the *callback* event handlers that are invoked when certain parsing events occur. These events (called SAX Events) provide all the information an XML-compliant application needs. We can leverage the work a SAX parser does by encoding the sequence of events.
UR	User requirement.

Table 1 - List of significant terms

1.4 References

[R.1] Dynamic content management on the ICA (FEP replacement)
Lot2_RPT_Dynamic content_management_on_the_ICA_v010.doc

[R.2] IDS Technical specification document
IDS_TSD1.00.doc

[R.3] CORDIS/IDS Functional Specification Document
IDS_R4_FSD1 00.doc

[R.4] Java Virtual Machine
http://www.java.com

[R.5] Apache Tomcat
http://jakarta.apache.org

[R.6] Oracle RDBMS
http://www.oracle.com

[R.7] Hibernate - Relational Persistence for Idiomatic Java
http://www.hibernate.org

[R.8] Lucene
http://jakarta.apache.org

[R.9] Cocoon
http://cocoon.apache.org

[R.10] Quartz scheduler
http://www.opensymphony.com/quartz

[R.11] JTidy - HTML to XHTML converter
http://jtidy.sourceforge.net/index.html

[R.12] ORO home page
http://jakarta.apache.org/oro/

2 ICADC introduction

The ICADC application replaces the phased off Fulcrum based FEP, providing a mechanism to generate dynamic content based on data stored in the ICA repository.

Based on the [R.1], this document will start with a detailed review of the FEP application, presenting then the ICADC application and the migration path in order to facilitate the reuse of information stored in templates and the query definitions (CALLERS) where possible.

2.1 Key architectural requirements

Since the ICADC application is a central part of the CORDIS infrastructure, replacing the FEP application, there were some key requirements that aim to facilitate its insertion into the ICA driven architecture while taking into account the future migration to the ICA2.

The main architectural requirements for the ICADC application were:

- "it must provide at least the same functions as the FEP application"
- "it must use the ICA content repository as its unique data source"
- "it must be based on a standard transformation technology such as XSL/XSLT"
- "it must take into account the future migration to the ICA2 and then use similar components where possible in order to facilitate future insertion into that content infrastructure"
- "it must re-use the existing template definitions or provide a simple conversion mechanism"

2.2 FEP architecture

The FEP application provided a mechanism to generate dynamic content using a Perl CGI script. The figure below presents its architecture:

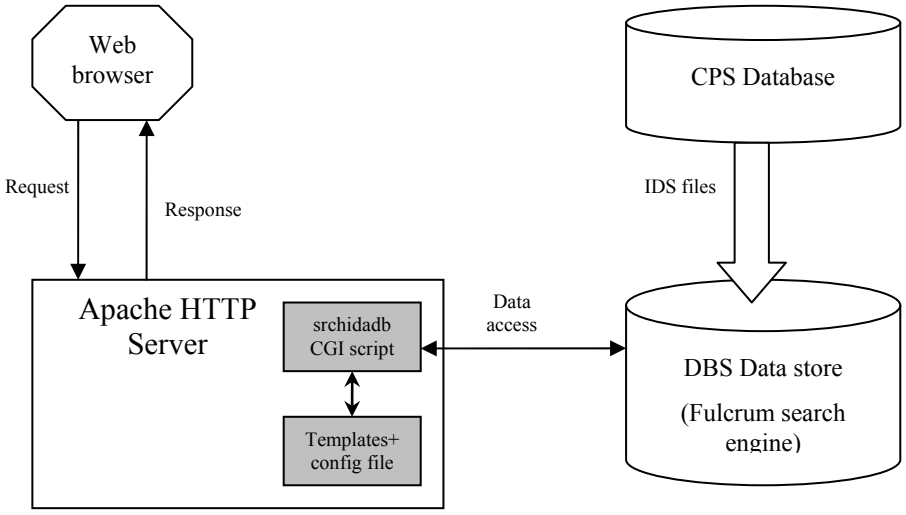

Figure 1 - FEP architecture

The components of this architecture are:

- Apache HTTP Server – the web server that hosts the CGI script
- Templates files – files containing proprietary tags that are interpreted by the CGI script
- Configuration file – specific configuration files for different type of requests
- DBS Data store – the search engine based on Fulcrum
- CPS Database – the database that builds the dissemination files
- IDS files – dissemination files are the input for indexing the Fulcrum search engine

A request to the CGI script triggers a set of actions:

- Parameter identification:
    1. identify parameters specified as part of the request
    2. identify parameters from the caller entry in the configuration file
    3. identify necessary parameters that where not specified in the previous steps from global section of the configuration file
- Template parsing, in order to identify the full list of the fields that need to be loaded
- Data loading, from the Fulcrum search engine
- Response sent back to the browser, based on template and the data loaded

Before any runtime call to the FEP application, the DBS is indexed with data parsed from dissemination files. A script from the CPS system produces the dissemination files. Each file has similar structure to the corresponding DBS table.

2.3 ICADC architecture

The ICADC application provides a powerful mechanism to generate dynamic content based on data stored in the ICA repository. The figure below presents its architecture:

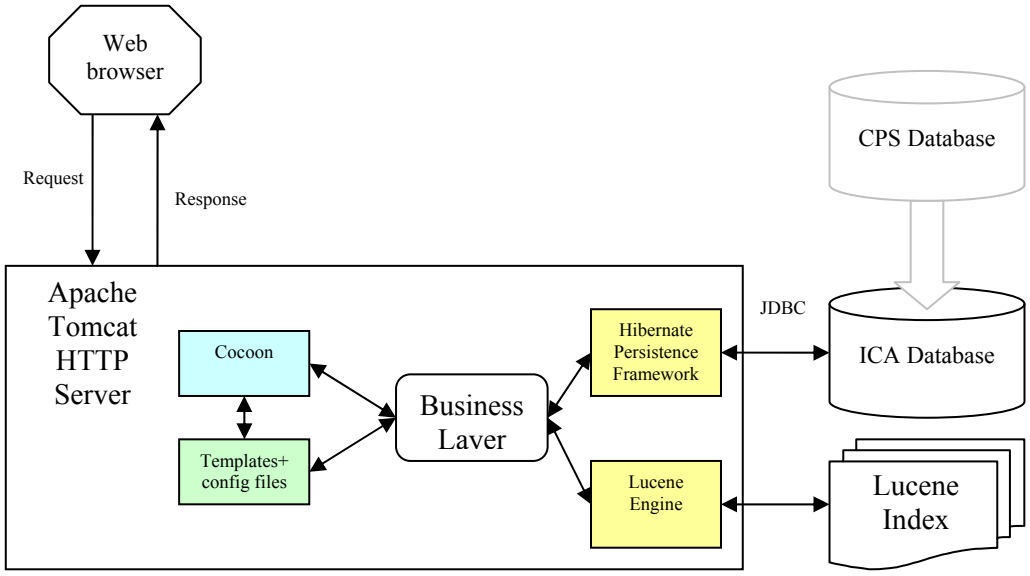

Figure 2 – ICADC architecture

The components of the new architecture are based on standard frameworks:

- Apache Jakarta Tomcat Web server – the Web container based on Java technology
- Apache Cocoon – a web development framework built around the concepts of separation of concerns and component-based Web development. This framework handles the logic of the application for HTTP request, configuration files and templates processing.
- Hibernate – object/relational persistence framework. It provides the "mapping" functionality,.
- Lucene – This component provides a search mechanism comparable to Fulcrum, but tightly coupled with other modules in order to provide an integrated solution.
- ICA Database – an Oracle database
- Lucene Search Index – this constitutes the "specific" data store for Lucene. This structure of data is stored in a file system.
- CPS – the content production data store.

2.4 XSL/XSLT transformation engine introduction

In order to achieve the full extensibility, the ICADC templating engine has the ability to use XSLT, improving different aspects of the quality of the solution when compared to the FEP application:

- separation of contents,
- code reuse,
- scalability.

The templates are basically XML well-formed files that contain custom tags. These templates are processed by the Apache Cocoon framework that drives the entire transformation cycle.

Apache Cocoon uses a XML dialect known as XSP (XML Server Pages) to drive the process. A XSP page is a XML file that contains embedded Java code.

Apache Cocoon reads the XSP pages using the Server Pages Generator (the standard Cocoon Generator). The latter executes the embedded logic and combines the output with the XML content stored in the rest of the file and processes it using a SAX based parser. The resulting SAX events are sent to the next component in the pipeline for processing.

The next step in the pipeline can be (the framework is flexible enough to accommodate different pipeline configurations) a XSLT transformation that transforms the XML output of XSP to another XML format.

The last step of the pipeline serializes the content in a XHTML format in order to be interpreted by the Web browser that acts as client.

The diagram below shows this pipeline configuration:

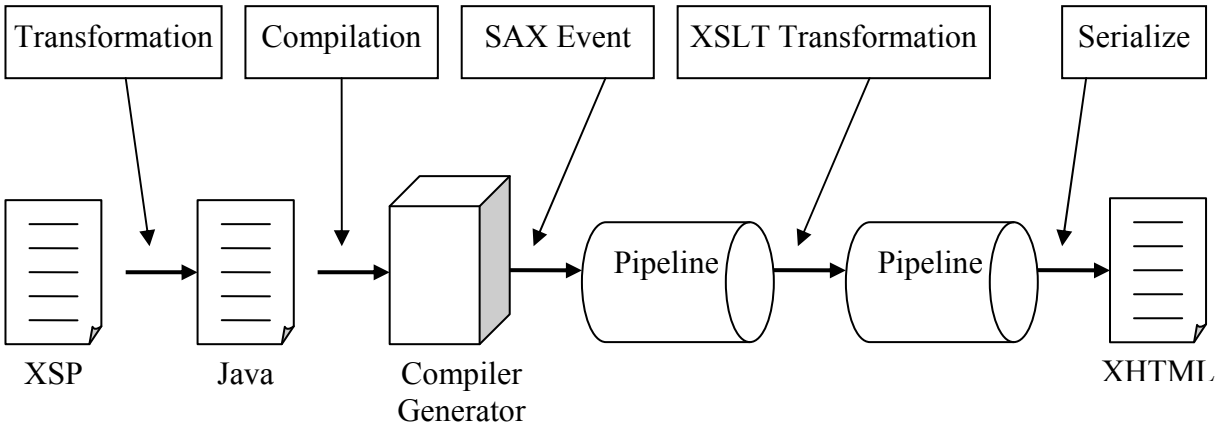

Figure 3 – Basic XSP processing flow

When a XSP is processed, it is actually transformed into a Java object. Apache Cocoon does this by creating a Java file, compiling it and executing it. Classically, the XSP page contains Java code inside. Because of that the code becomes extremely difficult to develop and to maintain. In order to avoid that, XSP offers the possibility to use the logic sheets. A logic sheet is a special kind of XSL style sheet, whose output is an XSP file.

An XSP logic sheet is a tag library that defines a set of custom XML tags which can be used within a XSP program to insert whole blocks of code into the file. Apache Cocoon comes with several predefined taglibs that can replace well-known artefacts of java coding logic.

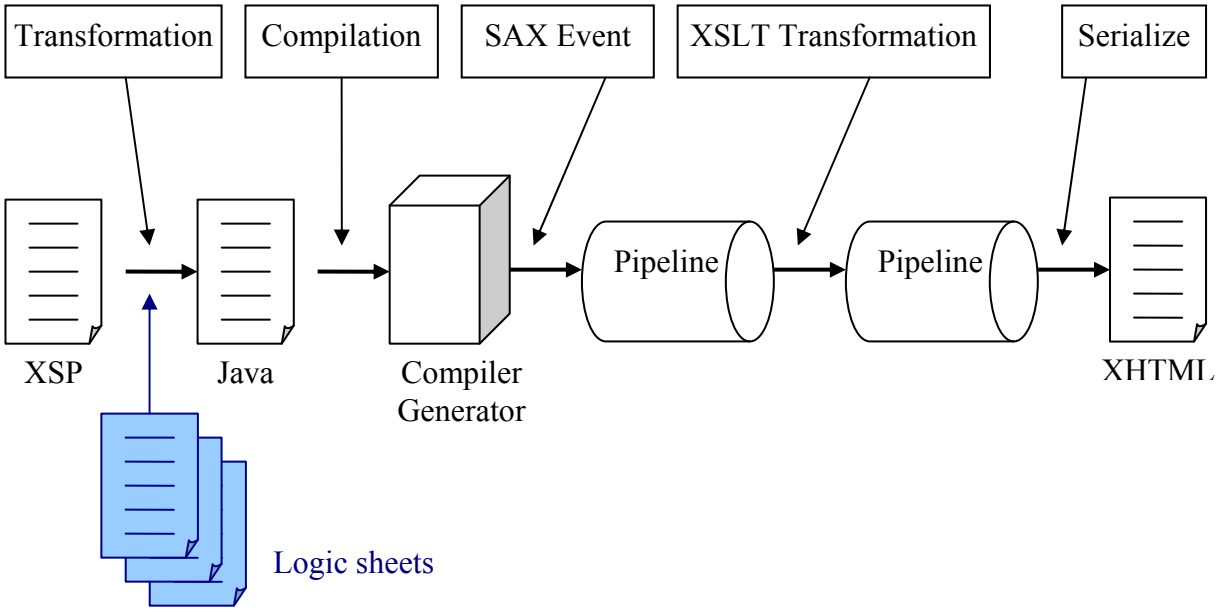

Figure 4 –Adding logic sheets on basic XSP processing flow

This approach has been used in order to develop custom style sheets that provide the required functionality to support the custom tags included in the ICADC templates.

During the transformation from HTML to XHTML, a human operator may move any part of HTML pages to XSLT file. For example, if the header or the footer needs to be common for a set of pages, then the XSLT solution is an optimal one to solve such situations.

In order to count on a solution that does not depend on any specific transformation implementation, the ICADC keeps these concept while using a custom implementation that is tailored to CORDIS needs. For the sake of compatibility we will keep the term XSP.

Below we can see an example of a template file in XML format:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<icadc:page
      xmlns:ica="http://www.cordis.lu/fep">
      <html>
      ……
      <body>
      ……

      <ica:val format="0" table="table1 " field="field1"/>

      ……
      </body>
      ……
      </html>
</icadc:page>
```

Figure 5 – Sample XML template

As we see, in order to make Apache Cocoon “understand” the custom ICADC tags it is necessary to specify that the XML document will use the “ICADC namespace” to localize the logic sheet domain.

The full details about the interaction between the ICADC custom tags will be provided in the section that describes the ICADC templating engine.

2.5 ICADC persistence layer and indexes

As shown in previous diagrams, the persistence layer is based on

- the ICA repository, basically used for document level views and on
- Lucene Indexes for result list views.

Lucene Indexes stores metadata providing a very fast search engine while providing all the required "summary" fields required for result lists generation.

2.5.1 Lucene concepts

The concepts of Lucene may be outlined as:

- An index contains a sequence of documents
- A document is a sequence of fields
- A field is named sequence of terms
- A term is a text string, keyed by field name

Documents are the primary retrievable units from a Lucene query. The fields that define the document have the same name with the ICA mapping referred in the templates. Each Lucene Index is referred by an ICA category and a language.

The properties of the fields in Lucene may be:

- Stored = non-inverted, content retrievable. The original text is available in the documents returned from a search.
- Indexed = inverted, searchable. This property makes the field searchable
- Tokenized = broken into tokens. The text added to the field is run through an analyzer and broken into relevant pieces. This has sense only for indexed fields.

Stored fields are handy for immediate access to the original text available from a search, such as a database primary key or filename. Stored fields can dramatically increase the index size, so these must be used wisely. Indexed field information is stored in an efficient manner, such that the same term in the same field name across multiple documents is only stored once, with pointers to the documents that contain it.

Lucene has predefined methods for possible combination of various attributes described earlier:

- Keyword -- Indexed and stored, but not tokenized. Keyword fields are useful for data like filenames, part numbers, primary keys, and other text that needs to stay intact as is.
- Text -- Indexed and tokenized. The text is also stored if added as a String, but not stored if added as a Reader. Our solution will always use the String accessing way.
- UnIndexed -- Only stored. Are not searchable.
- UnStored -- Indexed and tokenized, but not stored. Are ideal for text that needs to be searchable but need also to maintain the original text elsewhere or it is not needed for immediate display from search results.

In order to understand the solution, let's take some fields business that may appear in a document, to see what methods are need to be applied:

Field	Method	Stored	Indexed	Tokenized
RCN	UnIndexed	yes		
Title	Text	yes	yes	yes
Document detail	UnStored		yes	Yes
Country	Keyword	yes	yes	

Figure 6 – Example of fields and related indexing method

- The RCN field from any ICA category represents the primary key used to identify uniquely a document. It is not necessary to index it. It is used just to create the link from the results list page to the document level detail page.
- The Title field is necessary in order to display details in the results list about the document and for searching
- The Document detail is used just for searching. This field is not displayed in the results list.
- The Country field cannot be broken apart into tokens. For example if a country is "Czech Republic", then the match need to be exact on the complete name and not on pieces: "Czech" and "Republic". This method is necessary to eliminate any relative match.

These fields explained do not refer to a specific category. They simply illustrate a practical example for each kind of indexing method.

The Lucene Index is tightly related to the ICA. Initially the Lucene Index is created from ICA data retrieved using batch scripts.

The Lucene Index is stored in the same file system where the application is installed.

As we see in the diagram below, the batch script that will operate on the Lucene Index may be started directly, from command prompt or by the system scheduler. The indexing task is resource consuming and it is better to trigger it during the hours with less overall system load.

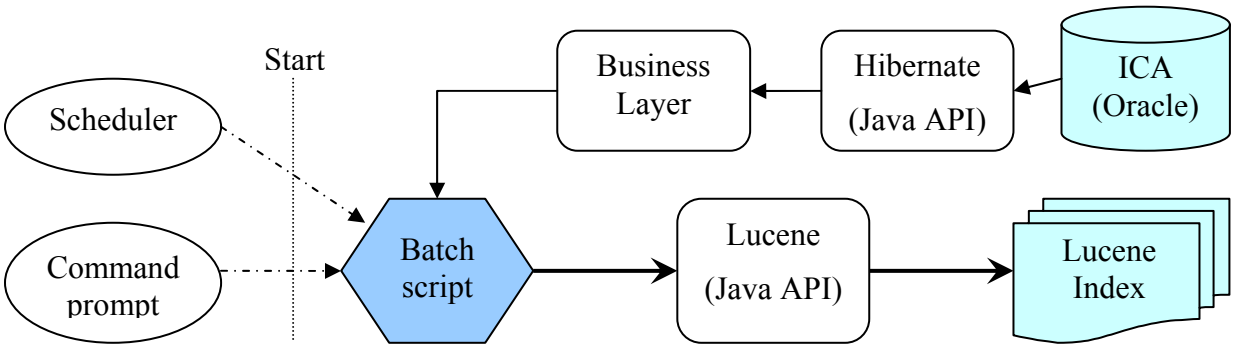

Figure 7 – Offline operations on the Lucene Index

Each category has a separate index for each language. If the category has only one language available and it is not specified then, this is considered to be default English.

A configuration file is used to store, the location and the name of indexes, function by languages and the fields that defines the index documents for each category. Below we can see an example:.

```
<indexes>
      <global>
            <location val=”/var/fep/indexes”/>
      </global>
      ……….
      <index category="TABLE1" >
            <languages>
                  <language name="EN" >
                        <location val="/opt/table1/en" />
                  </language>
                  <language name="DE"/>
                  ……..
            </languages>
            <fields>
                  <field ica="RCN" method="UnIndexed" />
                  <field ica="TTL" method ="Text"/>
                  <field ica="DOC_DETAIL" method ="UnStored" />
                  <field ica="COUNTRY" method ="Keyword" />
                  ……..
            </fields>
      </index>
      ……….
</indexes>
```

Figure 8 – Example of an entry in the Lucene Index configuration file

The configuration file contains a global section that defines the default location for the root of the indexes structure. The path to an index is obtained default concatenating:

default_root_location + category_name + language name

If for a specific category or for a language from a category a different location is required, then it is possible to redefine the location for the specific case adding the *location* inside the *language* element.

The batch script that creates or updates the Lucene Index expects to retrieving data from the business layer that stays on top of Hibernate persistence layer.

The business layer must provide:

1. Full list of records for a category.

   Input:

   - category name
   - a language identifier.

   Output:

   - list of records, having the field names exactly the ones described in the Lucene Index configuration file for related category

2. List of records that need to be updated

   Input:

   - category name
   - a language identifier.
   - date of last update

   Output:

- list of records, having the field names exactly the ones described in the Lucene Index configuration file for related category

When the scripts create the full index from scratch, the batch script automatically creates the directory structure where the index is stored.

In order to update the index, the batch script expects to have an index that respects the structure of the documents field described in the configuration file. When a document from the Lucene Index needs to be updated, in fact it needs to be deleted and re-added.

2.6 ICADC templating engine in detail

The biggest challenges for this component are:

- provide the rich set of features previously available on the FEP application.
- minimize the migration effort from existing templates
- support an open XSL/XSLT based technology
- minimize the development effort when migrating to the ICA2

The templates have three major functionalities:

1. Show results list after search operation
2. Show document level detail
3. Show an error about empty results list

The third functionality is used in cases where there are empty results list for both first two functionalities.

The Apache Cocoon engine may allow functionality like MVC design pattern concept, but at higher level of flexibility.

The templating engine was be developed in two stages, in order to support the templates migration steps:

- First version provided complete support of the old FEP style tags, with a limited set of extra capabilities.
- Second version extended the functionality with features provided by XSL/XSLT technologies. For new templates, the engine can provide functionality immediately. For the old templates, a testing period and extra operator attention is required, because the migration of templates from HTML to XHTML may generate interface issues that need to be verified manually.

The base architectures for both modes of operation are exactly the same. The only layer that exposes a different behaviour is the presentation layer.

Apache Cocoon has a rich set of tools for publishing web documents, and while XSP and Generators provide a lot of functionality, they still mix content and logic to a certain degree. The concept of Action was created to fill that gap. Because the Cocoon Sitemap provides a mechanism to select the pipeline at run time, sometimes there is a need to adjust the pipeline based on runtime parameters, or even the contents of the Request parameter. Without the use of Actions this would make the sitemap almost incomprehensible.

In our case, an Action is the proper place for request logic processing. The Action does not produce any display data. In fact is the only component that may allow dynamically logic to choose the page that needs to be used for rendering.

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
   <map:components>
      <map:readers default="resource">
         <map:reader name="tmpl" src="ica.fep.cocoon.reading.TmplReader"/>
      </map:readers>
      <map:actions>
         <map:action name="fepLogic" src="ica.fep.cocoon.acting.FepLogicAction"/>
      </map:actions>
      <map:selectors>
          <map:selector name="viewSelector"
                src="ica.fep.cocoon.selection.ViewSelector"/>
      </map:selectors>
    </map:components>
……
<map:match pattern="fep/pipeline/*">
      <map:act type="fepLogic">
         <map:select type=”viewSelector”>
            <map:when test=”reader”>
                <map:read type=”tmpl” mime-type=”text/html” src=”{template}”/>
            </map:when>
            <map:when test=”XSPOnly”>
                <map:generate type=”serverpages” src=”{xspPage}”/>
                <map:serialize src=”html”/>
            </map:when>
            <map:when test=”XSPandXSLT”>
                <map:generate type=”serverpages” src=”{xspPage}”/>
                <map:transform src=”{xsltTemplate}”/>
                <map:serialize src=”html”/>
            </map:when>
            <map:otherwise>
                <map:read src="{emptyResultsPage}" mime-type="text/html"/>
            </map:otherwise>
         </map:act>
      </map:act>
</map:match>
```

Figure 9 – Example of sitemap configuration file

As we may see in the example above, the application engine can use both implementations in parallel. The migration may be done in small steps and without affecting the production version of the site.

The Action takes care of:

1. checking request parameters
2. identifying the operation that need to be executed
3. if the operation is to make a search, then
    a. reads the specific request parameters
    b. asks the Business layer to interrogate the Lucene layer
    c. gets the reference to the Hits (Lucene search operation results)
    d. identifies the template(s) that need to be processed
    e. prepares the request parameters
4. if the operation is a “show a document level detail”
    a. reads the specific request parameters
    b. asks the Business Layer to retrieve from Hibernate layer the data detail for requested object
    c. gets the reference to the document details

d. identifies the template(s) that need to be processed

e. prepares the request parameters

The second step after the Action is the view processing. This second step may have different views available. The selection of the one that need to be used depends by the level migration. This level is defined by a new parameter named *MIGRATION_LEVEL* in the specific CALLER configuration entry. The accepted values for this parameter are:

- *reader* – for The templates may be HTML format but the tags must be the XML compatible. This value is considered as default if the parameter does not explicitly appear in the CALLERS configuration file.
- XSPOnly –The templates are expected to be XSP (is supposed to be XML compatible) but without additional XSLT file for transformation
- XSPandXSLT- This parameter is expected to be used for the new developed templates. it offers extensibility beyond XSPOnlyand it's the recommended option.

As we may see from the sitemap configuration showed as example in the Figure 15, the first option is to use a Reader in order to provide intermediate testing facilities. This kind of component offers direct manipulation of templates and skips the XML features but is necessary to see that the database migration was done successfully and the request parameters and the base logic of templates may be handled successfully.

The second option showed in the example from the Figure 15 proves that the migration of the old template was done in a fully functional way.

The logic of templates is included in the Action handler and it is the same for all the steps of the migration. In this way, the effort to support different forms of templates is reduced to a minimum. Also, the testers are able to identify correctly the source of any error, depending by the way of how the interface is implemented.

As we know, the templates engine is capable of generating multiple views of the stored data. The type of view that needs to be shown is specified by a set of parameters that may be identified from the request parameters or from the specific CALLER entry in the configuration file. The templates engine uses only one entry to handle both kind of views and related templates. In case of empty results list, or some other errors caused by a wrong interrogation for any of these pages, then the engine is automatically redirected to the template that shows the empty results. This is supposed to be a template, but it doesn't have any special tag inside. Because of that it may be read as a simple text/html page. It does not require special migration work to be done, but, if for any reason it will be necessary in the future, a mechanism may be added to support custom tags.

2.6.1 Results List templates

The first action that a user may do using this engine is to make a search and to obtain a results list. The starting point for such an action may be a static page that contains a form where the user may have possibility to enter and to organize different parameters for the search. Another way is to have a predefined link from a page with hard coded parameters. In both situations the "CALLER" parameter must be specified.

Each set of templates is strictly related to the related CALLER entry that has at least the next parameters:

- Table name – defines the category
- Template path – defines the path to the template

Having the category and the path to the templates the engine has the minimum information to identify the templates and to render it. As it was explained in the previous chapter, the interpretation is made inside the Action and the data is prepared on the request as reference. The view layer needs only to get reference to the specific hits object and to extract data for rendering.

In the below we can see a standard set of tags that may appear in results list templates. In the example XSP format is used, with the specific namespace. The same format that is supposed to be the standard for the new templates.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsp:page
      xmlns:xsp="http://apache.org/xsp"
      xmlns:xsp-request=http://apache.org/xsp/request/2.0
      xmlns:ica="http://www.cordis.lu/fep">
      <html>
      <body>
      ……
      <ica:totaldocs>
      ……
      <ica:results table=”table1”>
      ……
      <ica:body>
          ……
          <ica:seqno/>
          ……
          <ica:val format="0" table="table1" field="field1"/>
          ……
          <ica:val format="1" table="table1" field="field2"/>
          ……
          <doclink/>
          ……
      </ica:body>
      </ica:results>
      ……
      <ica:prvgroup/>
      <ica:nxtgroup/>
      ……
      </body>
```

Figure 10 – Basic example for Results List templates

As we see in the previous example, the structure of the custom tags is pretty simple.

- TOTALDOCS tag is used to display number of documents found.
- RESULTS tag is used to prepare the data to be assigned for the loop that will be start with BODY tag. The elements inside the BODY represent the detail that needs to be shown for each record founded by the search operation. For each of record, there is supposed to be shown the position in the results list (SEQNO), value for each field (specified in VAL tag) and link to the document detail (link created with by the DOCLINK tag).
- The other 2 tags PRVGROUP and NXTGROUP provide the link to the previous or next page with set of records for the current results list.

The table name and the fields names specified in the template need to match exactly the name of the ICA mapping described in the table Lucene Index configuration file as described before. Only the mappings described in this configuration file will be implemented in the Lucene Index and will be available for searching or results list rendering.

2.6.2 Document Detail templates

When a document detail view is requested, this kind of template is used. The starting point for this kind of page may be a link from the Results List page or from a static link.

In order to show a page the following parameters are required:

- Category name – identified from specific CALLER entry, specified on request
- RCN – the unique ID of a record
- Language – if this is missing, then English is considered default

Like for the Results List, the path and the identifier for the templates are also necessary, but these are supposed to be found in the specific entry in the CALLER configuration file.

Some of the tags used for the Document Detail templates are used in order to show the position in the Results List or the link to the next document in the Results List.

In the Figure below a standard set of tags that may appear in document details templates is shown.. In the example the XSP format is used, with the specific namespace, the same format that is standard for the new templates.

The document details usually show only the specific value for simple fields of the category entry. For some situations, it is possible to have a list of records related to the current record. In order to obtain this list, a special tag PERGROUP defines the relation between the base category and the sub-category used to provide the set of related records.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsp:page
      xmlns:xsp="http://apache.org/xsp"
      xmlns:xsp-request=http://apache.org/xsp/request/2.0
      xmlns:ica="http://www.cordis.lu/fep">
      <html>
      <body>
      ……
      <ica:docno>
      ……
      <ica:val format="1" table="table1" field="field1"/>
      ……
      <ica:val format="0" table="table1" field="field2"/>
      ……
      <ica:pegroup slaveTable="table2" slaveField="field10"
                   masterTable="table1" masterField="field3">
      ……
      <ica:body>
      ……
          <ica:seqno/>
      ……
          <ica:val format="0" table="table2" field="field4"/>
      ……
          <ica:val format="1" table="table2" field="field5"/>
      ……
          <doclink/>
      ……
      </ica:body>
      </ica:pergroup>
      ……
      <ica:prvdoc/>
      <ica:nxtdoc/>
      ……
      </body>
      </html>
</xsp:page>
```

Figure 11 – Basic example for Document Detail templates

As we see from the example shown in Figure 17 a document detail template may show the simple properties for a document and also a list of related records from another subcategory.

Like the Results List templates, the VAL tag shows the specific value of the field.

The position of the document in the Results List that creates the link to this view is provided by the DOCNO tag.

The PERGROUP tag prepares the list of related records from the subcategory. The rule is that the field from the slave subcategory table needs to match with the field from the master table. The master table is the table identified by the category of the Document Detail. The behaviour of this subcategory list functionality is very similar with the RESULT tag from the Result list templates. All the fields are cycled using the BODY field as delimiter.

The tags PRVDOC and NXTDOC create the link to the previous and the next document in the Results List.

2.7 Migration path

Migration of templates and configuration files (CALLER) from the FEP application to the ICADC needs to take into consideration the following aspects:

- Functionality must be compatible
- The migration effort must be minimized

In order to do that, it is necessary to re-use as much as possible from the interface described in the template files. A mechanism has been implemented in order to "understand" the old templates and migrate them in a new flexible set of tags, based on XML format. The migration has to be done in steps, with an intermediate level, described later.

Also, the database used for input data needs to provide access without modification. The CPS and ICA databases are considered to have a similar structure and that will not produce many modifications in the template tags attribute structure.

The old FEP templates looks like normal HTML pages, but inside have custom templates/tags identified by character sequences "~#" at the beginning and "#~" at the end.

Apache Cocoon is a publishing framework with strong foundations in XML-based server-side web application frameworks. The version 2 of the Cocoon introduces the concept of pipeline to handle requests, each component on the pipeline specializing on a particular operation. In order to use the full power of this framework it is necessary to transform the old templates from HTML format in XHTML format.

The old custom tags have references to the DBS. These references need to be migrated to be compatible with the ICA database. This step requires extensive testing.

Considering this, the migration has 2 main steps:

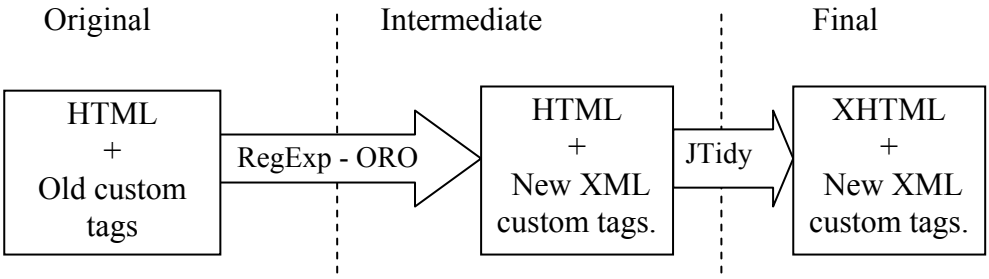

Figure 12 – Migration steps

First step involves:

- Transformation of the old "~#" and "#~" delimited tags to the new tags based on XML notation
- Update of formatter parameters
- Update of database referrers
- Verification of persistence layer implementation
- Verification of URL matching implementation
- Verification of tags handling implementation
- Verification of templates that need to be migrated

Second step involves:

- Transformation of templates to be XHTML compatible
- Extend normal functionality with XSLT if possible

The tools that are used for the template migration process are based on standard Cocoon components.

ORO is used especially for the matchers – there is possible to define complex rules for identifying correct pipeline to execute.

JTidy is used as a generator, to create automatically XHTML from any HTML input (file, link, etc).

The migration of templates has to be done offline in order to minimize the response time of the framework on requests.

2.7.1 Old custom tags meaning and replacement format

The first step involved in migration is parsing the old templates (based on a HTML structure + old custom tags), extracting the old tags and replace them with new XML compatible tags. The tool that will do that operation will be based on ORO libraries that provide regular expression functionality.

In general the old custom tags don't have many parameters, the functionality they provide being atomic. The new XML format of the custom tags will contain the ICA namespace in order to be prepared for the second step. The best way to see how these tags will be migrated is to look over some of them.

In the results list templates, the most important ones are:

TOTALDOCS

Old format: ~#TOTALDOCS#~

New format: <ica:totaldocs/>

Description: when interpreted, this tag will reveal the number of total results founded

RESULTS

Old format: ~#RESULTS EN_CONT#~ and ~#/RESULTS EN_CONT#~

New format: <ica:results table="EN_CONT"> and </ica:results>

Description: defines the starting of the results list loop. The additional parameter defines the table from which will be extracted data.

Observation: the previous table name will be migrated from the old Fulcrum reference to the new

BODY

Old format: ~#BODY#~ and ~#/BODY#~

New format: <ica:body> and </ica:body>

Description: Mark the code that will be cycled in order to show one record from the results list, started with RESULTS

SEQNO

Old format: ~#SEQNO#~

New format: <ica:seqno/>

Description: the tag is replaced by sequence number (order) of the record in the set of retrieved records.

VAL

Old format: ~#VAL 6 EN_NEWS.EN_RLTNS LNF=NWS LNC=NEWSLINK_EN_C#~

New format: <ica:val format="6" table=" NEWS" field=" RLTNS" lnc="NEWSLINK_EN_C" lnf= "NWS" />

Description: put the field value of a record from the results list. The parameters that appear in this tag are:

- format – specify how will be formatted the value of the field
- table – specify the table from where the information is loaded (may not be present)
- field - specify the field
- lnc – defines the caller that will be used to create a link (may not be present)
- lnf – defines the field that will be used to create a link (may not be present)

If the lnc and lnf are present, then the tag will create a link to another location. Technical, the table defined in the related caller and the field used to create the link will create a join relation with the main table and main field in order to identify the label and the location of the link.

Observation: The table name and the field name will be migrated from the old reference to the Fulcrum engine to the new mapping related to ICA database.

DOCLINK

Old format: ~# DOCLINK #~

New format: <ica:doclink/>

Description: provides link to the document detailed, referred by the current record from the results list.

PRVGROUP

Old format: ~# PRVGROUP #~

New format: <ica:prvgroup/>

Description: provides link to the previous set of records from the results list if the current set is not the first.

NXTGROUP

Old format: ~# NXTGROUP #~

New format: <ica:nxtgroup/>

Description: provides link to the next set of records from the results list if the current set is not the last.

The other important kind of templates is the one that describes the document details. Excepting the VAL tag, which has the same meaning, the other ones are totally new ones:

DOCNO

Old format: ~#DOCNO#~

New format: <ica:docno/>

Description: the tag is replaced by sequence number (order) of the record in the set of retrieved records.

PERGROUP

Old format: #~PERGROUP table2.field10?table1.field3#~

and ~#/PERGROUP table2.field10?table1.field3#~

New format: <ica:pegroup slaveTable="table2" slaveField="field10"

masterTable="table1" masterField="field3">

and </ica:pergroup>

Description: This tag defines a loop that iterates over a set of records obtained from *table2,* where the value of table1.*field3* equals to the value of table2.*field10.* The *table1* is detailed in current template.

Observation: The table name and the field name will be migrated from the old reference to the Fulcrum engine to the new mapping related to ICA database.

PRVDOC

Old format: #~PRVDOC~#

New format: <ica:prvdoc/>

Description: The tag is replaced by a hyperlink to the previous record of the set (in the corresponding result list that was used to access the current record).

NXTDOC

Old format: #~NXTDOC~#

New format: <**ica:nxtdoc**/>

Description: The tag is replaced by a hyperlink to the next record of the set (in the corresponding result list that was used to access the current record).

For both type of templates, there may appear some extra tags that have independent functionality:

PASSVAR

Old format: ~#PASSVAR: PGA#~

New format: <ica:passvar identifier="PGA" />

Description: replace the tag with the value defined by identifier, founded on the request.

FILELINK

Old format: #~FILELINK filename~#

New format: <ica:filelink identifier=”filename”/>

Description: The tag will be replaced with the content of the file that has been specified in the identifier attribute.

NONE

Old format: #~NONE~#

New format: <ica:none />

Description: The tag is migrated because of historical reason and it will be replaced by an empty string.

2.7.2 Database migration

Some of the tags described in the previous section are used as parameters identifiers for database tables or fields. The old FEP templating engine used references to the Fulcrum DBS database. The ICADC engine uses a relative mapping, not directly to a table name or a field name.

In order to understand the database mapping used to migrate to the new DB, it is necessary to identify the major databases or information structures from the system and the actual data transferring steps:

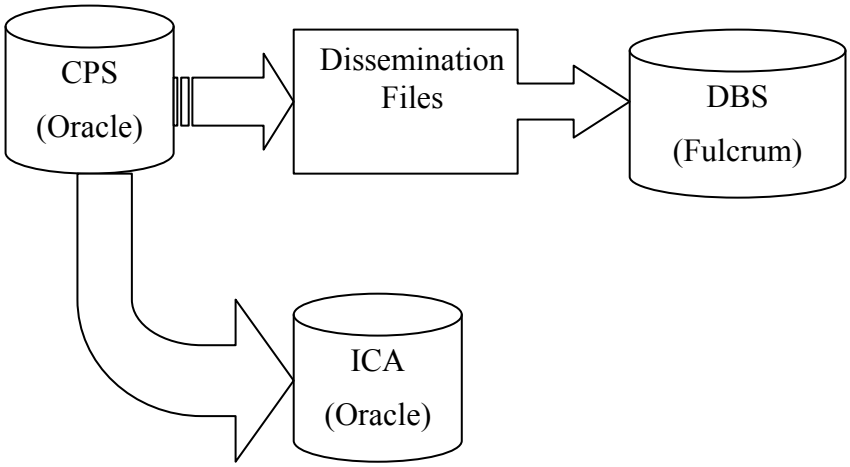

Figure 13 – Database relationships and data transfer

Where:

- CPS – the initial Oracle database used for updating operations.
- DBS – Search Engine Database developed on Fulcrum. The old template engine uses this database. The structure of information is accessible via limited SQL, for example: cannot join between tables.
- Dissemination Files – files generated from CPS to be input for DBS
- ICA – the new database, updated from the same initial CPS database. This is the database that is used by the ICADC application.

In order to update the database references from DBS to ICA, it is necessary to understand the intermediate steps of mapping, starting with the FEP templates mappings references (see Figure 4):

- Templates to DBS. This creates the list of mappings that are necessary to see the ICA correspondence
- DBS – Dissemination Files
- Dissemination Files – CPS
- CPS - ICA

The syntax of the database references from the old templates has a simple format:

TABLE_NAME.FIELD_NAME

This notation from the FEP templates is referred to the old simple Fulcrum flat tables and could not be reengineered to use directly the new ICA database because of the complexity behind of any entity model data.

In order to allow database access, a business layer that offers data retrieval using a similar syntax (to the old DBS reference) will hide the mapping logic.

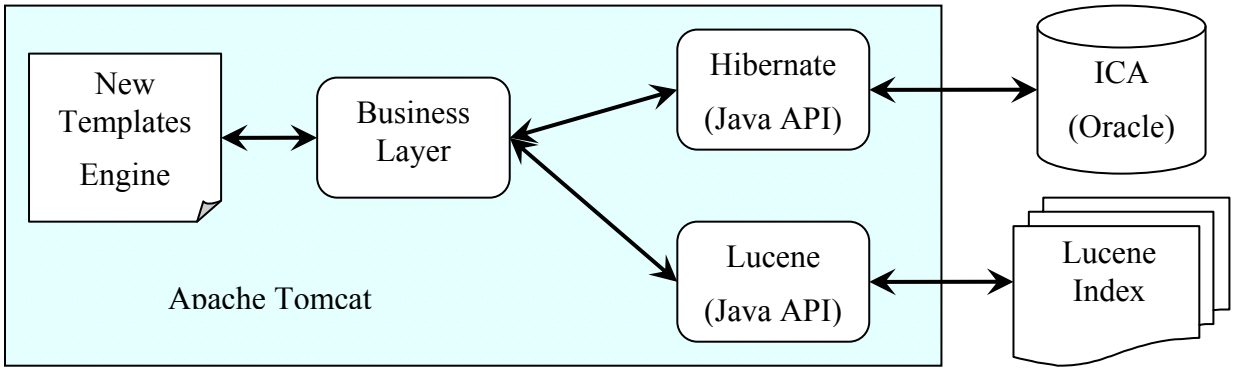

Figure 14 – New engine database access

As seen above shows, the new ICADC templating engine uses frameworks in order to access stored data. Comparing with the old Fulcrum engine that uses for both search and full data retrieval the same source, the new engine detaches these 2 as follows:

- Lucene for fast search functionalities (Lucene creates a proprietary index structure that is persisted by the file system)
- Hibernate to access ICA - Oracle RDBMS structure.

The business layer hides specific logic implemented by the Lucene and Hibernate and allows data access functionally similar with the old Fulcrum mappings in order to minimize the template migration effort.

The tool used to update the templates will use a XML configuration file in order to translate the old Fulcrum reference with a new one, ICA based. The differences between the new mapping and the old one are related to language interpretation/optimization. The language is no longer be a part of the category name. The new engine “knows” automatically how to retrieve data based on the general category name and the language.

The root element for this file is *categories*. For every category there is a *category* element that has three attributes:

- dbs – the name of the old DBS category
- ica - the name of the ICA category
- language – the identifier of the language for ICA category related to the old DBS reference

For every *category* element there is a mapping for each field, inside elements with the name *field* with the attributes:

- dbs – the name of the old DBS field
- ica - the name of the ICA field

```
<categories>
      <category dbs="EN_TABLE1" ica="TABLE1" language=”EN”>
            <field dbs="EN_RCN_A" ica="RCN_A" />
            <field dbs="EN_TTL" ica="TTL" />
            <field dbs="EN_DETAIL" ica="DETAIL" />
            <field dbs="EN_CAT_A" ica="CAT_A" />
            <field dbs="IMAGE_URL" ica="IMAGE_URL" />
            ……
      </category>
      <category dbs="DE_TABLE1" ica="TABLE1" language=”DE”>
            <field dbs="EN_RCN_A" ica="RCN_A" />
            <field dbs="EN_TTL" ica="TTL" />
            <field dbs="DE_DETAIL" ica="DETAIL" />
            <field dbs="DE_CAT_A" ica="CAT_A" />
            <field dbs="IMAGE_URL" ica="IMAGE_URL" />
            ……
      </category>
      ……
</categories>
```

Figure 15 – DBS-ICA mapping configuration

As seen above, an old *EN_TABLE1* DBS table name will be migrated to a *TABLE1* category. But also an old *DE_TABLE1* DBS table will be migrated to the same *TABLE1* category in the ICADC application. In order to identify the interface language, the new system will try to localize it in different contexts, prioritizing according to the following rule:

1. request parameter UPL
2. the specific CALLER entry in the configuration file for UPL parameter
3. Mapping configuration file DBS to ICA
4. global section of the CALLERS configuration file
5. If the UPL is not defined in any of the previous contexts, the interface language is considered to be English (EN)

2.7.3 Migration of the CALLERS configuration file

The FEP templating engine used a configuration file to maintain specific settings for each possible caller and a global section that keeps the default values for the case when the specific caller parameters are not specified.

In this configuration file there is a huge list of callers. For the ICADC engine, only a subset from the initial list needs to be migrated dynamically. Considering this, only the callers that need to be migrated will appear in the new configuration file. In the same time with this operation, the files that are related to the configuration caller entry will be extracted in a new structure and will be prepared to be used to minimize the space and the complexity of FEP’s files structure. The files that will be migrated are:

- Starting point search forms (if available)
- Templates for results list
- Templates for document level details
- Files referred by FILELINK tag

Let’s take a sample entry in the old CALLERS configuration file:

```
[MSS_NEWS_FR_FR]
TABLENAME=FR_NEWS
TEMPLATEPREFIX=MSS/FR/FR/
#SEARCH_PAGE=xxxx
SEARCH_TYPE=advanced
ACTION=R
DOC=1
RECORDS_DISPLAYED=10
RL_TMPL_TERM=FR_NEWS
LANGUAGE=FRENCH
DOC_TMPL_TERM=FR_NEWS
QM_EN_PGA_A=MS-FR C
USR_SORT=EN_QVD_A CHAR DESC
.............
```

Figure 16 – Caller entry in the old configuration file

This entry will be migrated in a XML format, taking as main element *CALLER* with name attribute the name of the caller from the old configuration file. The child elements will use the name of the key from the old configuration file.

```
<CALLER name="MSS_NEWS_FR_FR">
      <TABLENAME>NEWS</TABLENAME>
      <TEMPLATEPREFIX>MSS/FR/FR/</TEMPLATEPREFIX>
      <ACTION>R</ACTION>
      <RECORDS_DISPLAYED>10</RECORDS_DISPLAYED>
      <RL_TMPL_TERM>FR_NEWS</RL_TMPL_TERM>
      <LANGUAGE>FRENCH</LANGUAGE>
      <DOC_TMPL_TERM>FR_NEWS</DOC_TMPL_TERM>
      <QM_PGA_A>MS-FR C</QM_PGA_A>
      <USR_SORT>QVD_A CHAR DESC</USR_SORT>
      ……
</CALLER>
```

Figure 17 – Caller entry in the new configuration file

During the migration process, only the parameters that are still necessary will be migrated. For example SEARCH_TYPE refers to an older search method that was replaced in time by a more advanced engine.

Another important thing is that the reference to the related table for this caller will be migrated, based on the same mechanism described in the previous section. Also, will be migrated the other fields that may be references to the tables or fields (For example, EN_QVD_A from USR_SORT parameter will be migrated to QVD_A).

The commented field from the previous file will not appear in the new one. (`#SEARCH_PAGE=xxxx`). The global section from the configuration file will be migrated in a similar fashion like any other caller entry, but the main element will be named *GLOBAL*.

2.7.4 Migration of HTML page code to XHTML

JTidy is a Java port of HTML Tidy, a HTML syntax checker and a printer. It can be used as a tool for cleaning up malformed and faulty HTML. This parser checks the validity of the HTML code input by end-users and automatically tries to correct it. JTidy reads through the input file and if it finds any mismatched or missing end tags it corrects them and outputs a well-formed XML document. JTidy won't generate a cleaned up version when there are problems that it can't be sure of how to handle. This tool may be used to automate the

migration, but it will request operator attention. During the migration this tool generates errors or warnings, depending on the situation. These events need to be analyzed by somebody in order to see if the migrations were performed successfully.

A few examples how JTidy works:

- Missing or mismatched end tags are detected and corrected

  <h1>heading

  <h2>subheading</h3>

  It will be mapped to

  <h1>heading</h1>

  <h2>subheading</h2>

- End tags in the wrong order are corrected

  * here is aspecial paragraph.*

  It will be mapped to

  * here is a special paragraph. *

- Recovers from mixed up tags

  <h1>heading</h1>

  * new paragraph bold text*

  * some more bold text*

  It will be mapped to

  <h1>heading</h1>

  * new paragraph bold text *

  * some more bold text *

  - Getting the <hr> in the right place:

    <h1><hr>heading</h1>

    <h2>sub<hr>heading</h2>

  It will be mapped to

  <hr>

  <h1>heading</h1>

  <h2>sub</h2>

  <hr>

  <h2>heading</h2>

Adding the missing "/" in end tags for anchors:

<a href="#refs">References<a>

It will be mapped to

```
<a href="#refs">References</a>
```

- Missing quotes around attribute values are added
- Unknown/proprietary attributes are reported
- Tags lacking a terminating '>' are spotted

The major limitationsof JTidy are:

- It has limited support for XML
- Cannot recognize CDATA section
- Cannot recognize DTD subsets

These limitations will not reach the FEP templates, considering that the old HTML templates files are simple and without advanced tags, but this tool even if it is very useful, may generate design issues due the historical HTML interpreters and require operator attention. Because of that, extensive testing is envisaged.

2.8 Migration and testing

The migration from the FEP, as described before, cannot be done directly to the final standard of templates. If the engine will be tested simultaneously for all aspects, then there will be a risk to not identify correctly the source of an error. In order to avoid that, a two step procedure has been elaborated:

The first step will allow testing of components that replace the functionality of the old ones:

- Migration of database references from DBS to ICA
- Custom tags migration (from old format to new format)
- Configuration file migration
- Request parameters interpretation for Document Detail
- Direct access to the database for Document Detail
- Lucene index creation
- Request parameters interpretation for Results List
- Search functionalities and results list processing
- Lucene index updating

The second step will allow testing of extensibility to new features:

- Templates transformation from HTML to XHTML
- Extensibility using XSLT
- Integration with other external components

After the first step of migration, the engine will provide a similar functionality to the FEP application. The proof that the system was migrated successful will be that the new engine will work exactly like the old system after the first step. This step may be done fully automatic and without operator attention.

The second step requires operator attention. The tasks isolated in this second step require modifications or improvements that cannot be done fully automatic. Each CALLER that was migrated successful will have the *MIGRATION_LEVEL* parameter configured to activate the extended functionalities.

For the new templates, the second level will be considered default. The first step is necessary only to test the migration from the old engine for old templates. In this way, the testing period will be reduced at minimum.

2.9 Future migration to ICA2

The current architecture of this engine is modular and allows future improvements. In order to preview the major improvement that is scheduled for the CORDIS architecture, the migration to ICA2, it's important to understand the development effort implied.

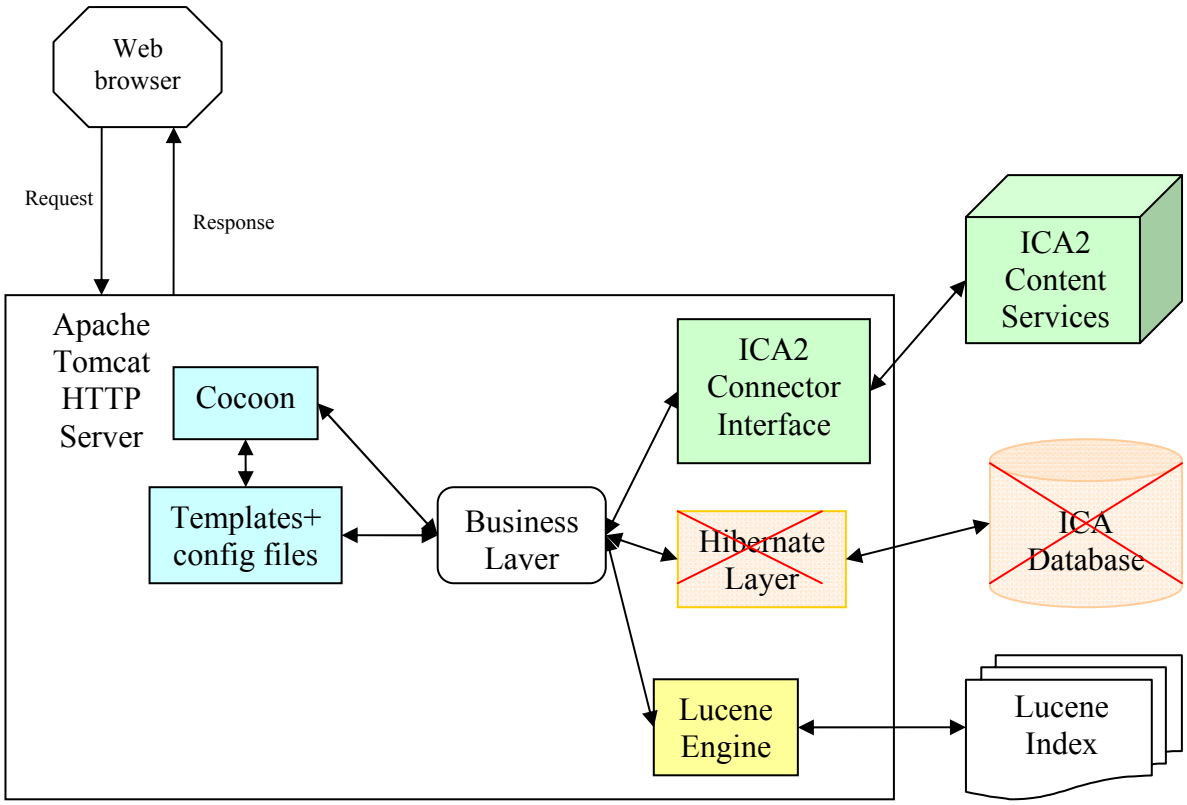

Figure 18 – ICA2 alternative architecture

In order to migrate to the new architecture, as seen above, only the database layer requires reengineering. The rest of the components will remain the same.