COMPASS
Functional specification

Christopher Kolstad

February 2009

Ovitas AS

www.ovitas.no

Table of Contents

1. Introduction....5
1.1 Introducing Compass....5
2. Interfaces....6
2.1 Knowledgebase – interface....6
2.1.1 importKB()....6
2.1.2 updateKB()....6
2.1.3 getExpansion(SearchObjectElementText, SearchConfig)....6
2.2 FullTextSearchInterface....7
2.2.1 addDocument()....7
2.2.2 deleteDocument()....7
2.2.3 doSearch()....7
2.3 CompassInterface....8
2.3.1 search()....8
2.4 LanguageToolInterface....8
2.4.1 getStem(s)()....8
2.4.2 getSpellingSuggestion()....8
2.4.3 getSpellingSuggestions()....8
3. Implementation....8
3.1 SearchObject....8
3.1.1 SearchObject....9
3.1.1.1 SearchObject()....9
3.1.1.2 AddTextSearch()....9
3.1.1.3 AddDateSearch()....9
3.1.1.4 AddNumberSearch....10
3.1.2 SearchObjectElement....10
3.1.2.1 SearchObjectElement....10
3.1.2.1.1 terms()....10
3.1.2.2 SearchObjectElementText....11
3.1.2.3 SearchObjectElementDate....11
3.1.2.3.1 getStartDate()....11
3.1.2.3.2 getEndDate()....11
3.2 KnowledgeBaseInterfaceImpl....11
3.2.1 ImportKB()....12
3.2.2 UpdateKB()....13
3.2.3 GetExpansion() 13
3.2.4 SaveKnowledgeBase()....14
3.2.5 UploadTransitivePath()....14
3.2.6 ListKnowledgeBases()....15
3.2.7 UpdateRelationWeight()....15

3.2.8 UpdateRelationTypeWeight()....15
3.3 FullTextSearchInterfaceImpl....16
3.3.1 addDocument()....16
3.3.2 deleteDocument()....17
3.3.3 doSearch()....17
3.4 CompassInterfaceImpl....19
3.4.1 search()....19
3.5 LanguageToolsInterfaceImpl....20
3.5.1 getStem(s)()....20
3.5.2 getSpellingSuggestion()....20
3.5.3 getSpellingSuggestions()....21
4. Demo Web Application....22
4.1 Search box....22
4.2 Configuration toggling....22
4.3 Result presentation 22
5. Models....23
5.1 Use Case Model....23
....23
5.2 Domain Model....25
5.3 Class Model....26
6. Configuration....27
6.1 Full text search part....27
6.1.1 Full-text-search tag....27
6.1.2 Full-text-search-implementation tag....28
6.1.3 Params tag....28
6.1.4 Param tag....28
6.1.5 Document-fields tag....29
6.1.6 Fields tag....29
6.1.7 Indexer-implementation tag....29
6.1.8 Content-handler-implementation tag....30
6.2 Language tools part....30
6.3 Knowledge bases part....30
6.3.1 Knowledge-bases tag....31
6.3.2 Knowledge-base tag....31
6.3.3 Knowledge-base-implementation tag....32
6.3.4 Params tag....32
6.3.5 Param tag....32
6.3.6 Knowledge-spring-implementation tag....32
6.3.7 Context-files tag....33
6.3.8 Context-file tag....33
6.3.9 Params tag....33

6.4 Search bases part....34
6.4.1 Search-bases tag....34
6.4.2. Search-field-string tag....34
6.4.3 Search-field-dateinterval tag....35
6.4.4 Search-field-numeric tag....35
6.5 Result part....36
6.5.1 Result tag....36

1. Introduction

1.1 Introducing Compass

“Compass is an information retrieval solution based on a user defined knowledge model. Compass enables you to find relevant information in a quick and intuitive manner”. The Compass solution is based on a combination of traditional full text search and Topic Maps navigation. This combination produces precise and relevant hits to your search, maintaining the semantic space all the way. Compass represents a unique and revolutionary approach to search and information retrieval. Compass provides seamless integration between search and navigation. Hits can be grouped by search terms, topic types or association types. Topics are underlined in this grouping, so clicking on these links the user can further navigate in the semantic space.
Using Compass, the end user can get useful information related to a query term, even when this term does not occur in any document in the search domain. The model assists the information retrieval action in such a way that the user will get relevant hits without having deep knowledge of the domain terminology. Compass expands the query with similar terms and synonyms, and returns more hits that are semantically related to the query term. These hits are weighted, so it is possible to present the relevant hits only and ignore the non-relevant ones.

- Seven APIs makes up Compass, one for Import interaction, one for KnowledgeBase interaction, one for FullTextSearch interaction, one for LanguageTools, one for Search interaction, one for Result interaction and one for Compass.
- The KnowledgeBase API handles interaction with the KnowledgeBase.
- The FullTextSearch API handles interaction with whatever FullTextSearch Engine is being used. It should expose methods for performing searches, updating the index and adding/deleting documents from the index.
- The LanguageTools API should expose methods for stemming and spelling suggestions.
- The Import API handles KnowledgeBase files import, be it RDF/OWL or TM.
- The Search API provides simple solution for search in KnowledgeBases and indexed documents.
- The Result API helps get the search results.
- The Compass API is a single entry point to the other interfaces. It exposes one single method, search. Which starts by querying the KnowledgeBase for related topics to the search, then queries the FullTextIndex for the constructed query from the entered search and the related topics to the search. It then returns both the results from the FullTextSearch and the result from the KnowledgeBaseQuery for further processing by web applications.

2. Interfaces

2.1 CompassManager – interface

Should expose the following methods

2.1.1 createTopicQuery()

Create a TopicQuery, with this query can search in Knowledge bases.

2.1.2 createFullTextQuery()

Create a FullTextQuery, with this query can search in FullText indexes.

2.1.3 searchTree(TopicQuery)

Search in knowledge bases and the results are contained in tree format.

2.1.4 search(TopicQuery)

Search in knowledge bases and the results are contained in list format.

2.1.5 search(FullTextQuery)

Search in full text index.

2.1.6 search(FullTextQuery, FullTextTopicQuery)

Search in full text index with topic extension.

2.1.7 search(TopicQuery, FullTextQuery)

Search in knowledge base and full text index. If topic query search has result it will be converted to FullTextTopicQuery and use 2.1.6 search(FullTextQuery, FullTextTopicQuery) method for full text search.

2.1.8 getSpellingSuggestion(String)

Get spelling suggestion for the specified term.

2.1.9 getSpellingSuggestions(String)

Get spelling suggestions for the specified term.

2.1.10 getSpellingSuggestions(String, int)

Get spelling suggestions for the specified term. With second parameter can maximize the number of suggestions.

2.1.11 getStem(String)

Get stem of the term.

2.1.12 getDocument(String)

Get document details for document which specified with id parameter.

2.1.13 getAvailableSearchFields()

Get available search fields. Only can search in these fields in FullTextQuery.

2.1.14 getDefaultSearchConfig()

Get default search configurations.

2.1.15 getDefaultSearchFields()

Get default search fields. This is a set of the 2.1.13 getAvailableSearchFields() result.

2.1.16 getAvailableUploadTypes()

Get available upload file types. Practicality the importable knowledge base types.

2.1.17 newInstanceKowledgeBase(KnowledgeBaseType)

Create new KnowledgeBase instance. In the import process will be used this instance.

2.1.18 importKnowledgeBase(String, String, KnowledgeBase)

Import knowledge base and start knowledge base storing process.

2.1.19 getImportedScopes()

Get all available scopes from knowledge base which is under importing.

2.1.20 filterImportedKnowledgeBase(Collection<Long>)

Filer knowledge base is under importing. This is the second step of knowledge base storing process.

2.1.21 getImportedRelationTypes()

Get relation types from knowledge base is under importing.

2.1.22 storeImportedKnowledgeBase(Collection<RelationTypeSetting>)

Store imported knowledge base. Before store it all relation types weight will be updated with specified values in the relationTypeSettings parameter. This is the final step of knowledge base storing process.

2.1.23 listKnowledgeBases()

List the descriptors of the available knowledge bases.

2.1.24 getKnowledgeBase(long)

Get the knowledge base descriptor of the knowledge base having the specified id.

2.1.25 getScopes(KnowledgeBaseDescriptor)

Get scopes from the specified knowledge base.

2.1.26 getRelationTypes(KnowledgeBaseDescriptor)

Get relation types from the specified knowledge base.

2.1.27 updateRelationTypes(Collection<RelationTypeSetting>, KnowledgeBaseDescriptor)

Update relation types weights in the specified knowledge base.

2.1.28 getFilesFromKnowledgeBaseImportDirectory()

Get files from the specified import directory. This settings come from configuration XML.

2.1.29 indexDocument(boolean, int, String, String)

Index documents under specified directory. The directory can be a single file then only it will be indexed. With the depth parameter can set the max depth of the indexing.

2.1.30 indexDocuments(String, String)

Index documents like 2.1.29 indexDocument(boolean, int, String, String), but the parameters are in a configuration file.

2.1.31 reReadConfig()

Read the configuration file again and refresh the managers.

2.2 FullTextSearch - interface

Exposes the following methods.

2.2.1 doSearch(FulltextQuery, FullTextTopicQuery)

Search in full text index with topic extension. The topic expansion can be null.

2.2.2 listAllContentHandlerType()

List all available content handler type.

2.2.3 indexDocument(boolean, int, String, String)

Index documents under specified directory. The directory can be a single file then only it will be indexed. With the depth parameter can set the max depth of the indexing.

2.2.4 indexDocument(String, String)

Index documents like 2.2.3 indexDocument(boolean, int, String, String), but the parameters are in a configuration file.

2.2.5 getDocument(String id)

Get document details for document which specified with id parameter.

2.3 FTSContentHandler – interface

Exposes the following methods.

2.3.1 getContent(File)

Return with the content of the file.

2.3.2 setFields(Collection<Field>)

Set the field whet we want to contain the content.

2.4 FTSIndexer – interface

Exposes the following methods.

2.4.1 index(Content)

Index the content.

2.4.2 closeIndexer()

Close the indexer. Use this only after index documents.

2.4.3 openIndexer(boolean)

Open idexer. If parameter is true then delete index repository and create a new else all content will be added to the existed repository.

2.4.4 getDocument(String)

Get the document from the index.

2.4.5 updateIndex(Content)

Update the indexed content with the new one. Two content is equls if theirs URI fields are same.

2.4.6. deleteDocument(String)

Delete document from index repository. The parameter is the document URI field because this field is unique.

2.4.7 doSearch(FullTextQuery, FullTextTopicQuery)

Search in full text index with topic extension. The topic expansion can be null.

2.5 QueryBuilder – interface

Exposes the following methods.

2.5.1 getCreatedQuery()

Get created query. This is the final step.

2.5.2 startCreateQuery()

Start create query. This is the first step.

2.5.3 addCriteria(FullTextFieldCriteria, FullTextTopicQuery)

Add FullTextFieldCriteria to the processing query. IF the FullTextTopicQuery contains either of the criteria terms then add its values to the query. The FullTextTopicQuery can be null.

2.6 KnowlegdeBase - interface

Should expose the following methods

2.6.1 getRelationTypes(KnwoledgeBaseDescriptor)

Get the relation types from specified knowledge base.

2.6.2 getScopes(KnowledgeBaseDescriptor)

Get the scopes from specified knowledge base.

2.6.3 listActiveKnowledgeBases()

Get all active stored knowledge bases. The knowledge base is not active while the store process is not finished.

2.6.4 deleteKnowledgeBase(KnowledgeBaseDescriptor)

Delete specified knowledge base.

2.6.5 storeKnowledgeBase(KnowledgeBase)

Save a new knwoledge base or update exist. The update means all element will be removed except relation types. Those relation types which not represented in the knowledge base instance will be deactivated.

2.6.6 getKnowledgeBase(long)

Get the knowledge base descriptor of the knowledge base having the specified id.

2.6.7 updateRelationTypeWeight(RelationTypeSetting, KnowledgeBaseDescriptor)

Modify the relation type's weights.

2.6.8 newInstanceKnowledgeBase(KnowledgeBaseType)

Create new knowledge base instance.

2.6.9 getExpansionTree(TopicQuery)

Search in knowledge bases and the results are contained in tree format.

2.6.10 getExpansionList(TopicQuery)

Search in knowledge bases and the results are contained in list format.

2.6.11 getFilesFromImportDirectory()

Get files from the specified import directory. This settings come from configuration XML.

2.6.12 setTopicNameIndexer(TopicNameIndexer)

Set topic name indexer. This is important object because we search topics with this object and the result topics will be extended.

2.7 TopicNameIndexer – interface

Should expose the following methods.

2.7.1 saveKnowledgeBase(KnowledgeBase)

Save knowledge base.

2.7.2 removeKnowledgeBase(KnowledgeBaseDescriptor)

Remove specified knowledge base.

2.7.3 getRelatedResults(Collection<String>, long, int, boolean)

Get the related results for the search terms.

2.8 LanguageToolInterface

Facilitates stemming and spelling suggestions.

2.8.1 getStem(String)

Stems the word passed in and returns the stemmed word

2.8.2 getStems(Collection<String>)

Stems the words passed in and returns the stemmed words

2.8.3 getSpellingSuggestion(String)

Get the highest ranked spelling suggestion for the word entered.

2.8.4 getSpellingSuggestions(String)

Get all spelling suggestions for the word entered.

2.9 ImportManagerInterface

The import manager can manage only one import process at once.

2.9.1 listAvailableImportPlugins()

List all available ImportManagerPlugin 2.10 ImportManagerPluginInterface instance.

2.9.2 importKB()

Start to import the imported knowledge base. Choose the responsible ImportManagerPlugin 2.10 ImportManagerPluginInterface and call the 2.10.1 importKB() method.

2.9.3 getScopes()

Get the scopes from the imported knowledge base.

2.9.4. filterImprtedKnowledgebase(Collection<Long>)

Filter the topics and relations with the specified scopes. Topics and Relations which doesn't be in one specified scope are deleted.

2.9.5 getRelationTypes()

Get the RelationTypes from the imported knowledge base.

2.9.6 getKnowledgeBaseModel()

Return with the model of the imported knowledge base.

2.9.7 getKnowledgeBaseDescriptor()

Return with the knowledge base descriptor of the imported knowledge base.

2.10 ImportManagerPluginInterface

2.10.1 importKB()

Import knowledge base from the specified File or URL or InputStream.

2.10.2 getScopes()

Get the scopes from the imported knowledge base.

2.10.4. filterImprtedKnowledgebase(Collection<Long>)

Filter the topics and relations with the specified scopes. Topics and Relations which doesn't be in one specified scope are deleted.

2.10.5 getRelationTypes()

Get the RelationTypes from the imported knowledge base.

2.10.6 getKnowledgeBaseModel()

Return with the model of the imported knowledge base.

2.10.7 getKnowledgeBaseDescriptor()

Return with the knowledge base descriptor of the imported knowledge base.

2.10.8 cleanUp()

Clean up after import process.

2.11 Manager – interface

All interface before that extends Manager interface. There should expose the following methods all of them except 2.5 QueryBuilder – interface.

2.11.1 setConfiguration(ConfigurationManager)

Set the configuration manager.

2.11.2 init(Properties)

Initialize the manager. After create a manager instance need to call this method. In the properties parameter contains all parameters from configuration XML.

2.12 Query – interfaces

2.12.1 TopicQuery – interface

2.12.1.1 addTerms(Collection<String>)

Add terms to the query.

2.12.1.2 addTerm(String)

Add term to the query.

2.12.1.3 addTermWithStem(String, String)

Add term with a stemmed term. The query search with the stemmed term.

2.12.1.4 getTermStemPairs()

Get the term stem pairs. It contains those terms too which add with 2.12.1.1 addTerms(Collection<String>) and 2.12.1.2 addTerm(String) in this way the pair key and value contains same string.

2.12.1.5 getSearchTerms()

Get search terms. This will be used for search.

2.12.1.6 createTopicCriteria()

Create new TopicCriteria instance.

2.12.1.7 getTopicCriterias()

Get created TopicCriteria instances.

2.12.1.8 isPrefixSearch()

Return true if it prefix search.

2.12.1.9 setPrefixSearch(boolean)

Set prefix search value.

2.12.1.10 getThresholdWeight()

Get threshold weight.

2.12.1.11 setTresholdWeight(double)

Set threshold weight value.

2.12.1.12 getMaxTopicNumberToExpand()

Get max topic number to expand value.

2.12.1.13 setMaxTopicNumberToExpand(int)

Set max topic number to expand value.

2.12.1.14 getHopCount()

Get hop count number.

2.12.1.15 setHopCount(int)

Set hop count value.

2.12.1.16 isTreeSearch()

Return true is this is tree search.

2.12.1.17 setTreeSearch()

Set tree search value.

3. Implementation

3.1 SearchObject

The inner representation the search expression. The SearchObject type store SearchObjectElements. This elements contains the search values, search value type, whichfield want u search in the indexer and the relation with the result. It means if the value wanted to search by title field and the result not contains the searching expression need to set the relation to NO.

Three SearchObjectElement implementation created. The first is SearchObjectElementText can use for search texts and strings. The second one for date interval search is

SearchObjectElementDate. Last one is the basic implementation of the SearchObjectElement.

It can use for all other searching terms.

3.1.1 SearchObject

3.1.1.1 SearchObject()

Input parameter	Explanation	optional
Boolean fastSearch	If true, then some plugin works otherwise.	No
Return value	Represented SearchObject	

If fast search enabled the SearchObjectElements relation always set to OR.

3.1.1.2 AddTextSearch()

Input parameter	Explanation	optional
String searchExpression	The search expression. It will be split to words, if want to search expression which contains more words place them between quotation marks.	No
SearchField field	Describe in which field want to search.	No
ConnectingType connecting	The relation type with the result.	No

FittingType fitting	Describe which query type will be created. PrefixQuery or WholeWordQuery.	No
MatchingType	The relation between the worlds in the expression.	No
Return value	void	

3.1.1.3 AddDateSearch()

Input parameter	Explanation	optional
Date startDate	The start date of searched interval.	No
Date endDate	The end date of searched interval.	No, but may be null.
SearchField field	Describe in which field want to search.	No
ConnectingType connecting	The relation type with the result.	No
Return value	void	

3.1.1.4 AddNumberSearch

Input parameter	Explanation	optional
Number value	The search value	No
SearchField field	Describe in which field want to search.	No
ConnectingType connecting	The relation type with the result.	No
Return value	void	

3.1.2 SearchObjectElement

It can make some transformation on the searching data. Example the

SearchObjectElementText split the search text to words. In generally you don't need to create or put data to SearcObjectElement object the SearchObject do that for you.

3.1.2.1 SearchObjectElement

3.1.2.1.1 terms()

Input parameter	Explanation	optional
Return value		
Set<T> terms	Return with the stored terms. The T is the terms type.	

3.1.2.2 SearchObjectElementText

Only have the same method as 3.1.2.1.1 terms(). The return value type is Set<String>.

3.1.2.3 SearchObjectElementDate

Have the same method as 3.1.2.1.1 terms(), but return value not sorted.

3.1.2.3.1 getStartDate()

Input parameter	Explanation	optional
Return value		
Date startDate	Return with the earlier date.	

3.1.2.3.2 getEndDate()

Input parameter	Explanation	optional
Return value		
Date endDate	Return with the later date if it null return with the current date.	

3.2 KnowledgeBaseInterfaceImpl

We want a basic implementation of this for now. A simple in-memory database or an xml file for parsing on startup will do just fine, we even have a finished dtd for this, we'll translate this to English and add the weighting on the relation type.

The xml file will contain relationship type definitions, and an entry for each topic, with each relation from that topic listed underneath it as children of the topic node. The getExpansion() method should then get all children from the topic matched by the topic passed in. If called with a hopcount of more than 1 should then try to find children of all of the children of the original topic for the second hop, and then children of all of the children of all of the original children for the third hop and so on for each new hop and as long as there was at least topic found in the expansion with a weight higher than the threshold.

We have three classes, Topic, Relation and RelationType.

A Topic has the following class signature
class

```
Topic {
        private String name;
        private List<String> alternativeNames;
        private Set<Relation> relations; // The relations we're part of
}
```

Relation has the following signature

```
class Relation {
        private Topic source;
        private Topic target;
        private double weight;
        private RelationType relationType;
}
```

```
class RelationType {
        private String relationName;
        private double weight;
}
```

The knowledgebase implementation is up to you, but it needs to be able to support getting topics from it based on just plain equality between the user entered search and a topic name, prefix matching of the user entered search and a topic name and fuzzy matching the user's search and a topic name.

3.2.1 getRelationTypesInKnowledgeBase()

Input Parameter	Explanation	Optional
KnowledgeBaseDescriptor	Specified the knowledge base.	No
Return Value		
List<TopicTreeNode> resultTopics	A list of the RelationTypes which are contained by the knowledge base.	

3.2.2 replaceKnowledgeBase()

Input Parameter	Explanation	Optional
KnowledgeBase	The instance of the knowledge base	No
Return Value		
void		

Save the knowledge base if it has existed then replace it.

3.2.3 saveAndUpdateKnowledgeBase()

Input Parameter	Explanation	Optional
KnowledgeBase	The instance of the knowledge base	No
Return Value		
void		

Save the knowledge base or if it has existed then replace all entities except the RelationTypes and update the RelationTypes.

3.2.4 uploadTransitivePath()

Input Parameter	Explanation	Optional
KnowledgeBaseDescriptor	Define the knowledge base.	No
int hopCount	The max transitive path length	No
Return Value		
void		

3.2.5 listKnowledgeBases()

Input Parameter	Explanation	Optional
Return Value		
List<KnowledgeBaseDescriptor>	All contained knowledge bases.	

3.2.6 getExpansionSpecialization()

Input Parameter	Explanation	Optional
Collection<SearchObjectElement Text>	The query to run against the knowledge base.	No
SearchConfig 1. Double ThresholdWeight 2. Int hopCount 3. Boolean fuzzyMatch	That contains the older version's parameters. 1. At which weight should the expansion stop 2. How many hops should be done 3. True only if we want the query against the knowledgebase to be done as a FuzzyQuery	No
KnowledgeBaseDescriptor	Define the knowledge base.	
Return Value		
Map<String, TopicSearchResultContainer>	A list of TopicTree containing all the topics found during the expansion in the root of each tree. Each tree contains all other topic which are satisfy the config	

	parameters. All edge in the tree are relation in knowledgebase.	

Finds the user entered search in the Knowledge base and perform expansion from it.

Weight of expanded topics are the product of the weight of the relations that have been followed to find the topic. i.e. finding a topic on the second hop where the first hop had a weight of 0.8 and the second hop had a weight of 0.7 gives a weight of 0.56. The weight must be above the threshold in order for the topic to be included in the result.

3.2.7 getExpansionGeneralization()

Input Parameter	Explanation	Optional
Collection<SearchObjectElement Text>	The query to run against the knowledge base.	No
SearchConfig 4. Double ThresholdWeight 5. Int hopCount 6. Boolean fuzzyMatch	That contains the older version's parameters. 1. At which weight should the expansion stop 2. How many hops should be done 3. True only if we want the query against the knowledgebase to be done as a FuzzyQuery	No
KnowledgeBaseDescriptor	Define the knowledge base.	
Return Value		
Map<String, TopicSearchResultContainer>	A list of TopicTree containing all the topics found during the expansion in the root of each tree. Each tree contains all other topic which are satisfy the config parameters. All edge in the tree are relation in knowledgebase.	

Finds the user entered search in the Knowledge base and perform expansion from it.

Weight of expanded topics are the product of the weight of the relations that have been followed to find the topic. i.e. finding a topic on the second hop where the first hop had a weight of 0.8 and the second hop had a weight of 0.7 gives a weight of 0.56. The weight must be above the threshold in order for the topic to be included in the result.

3.2.8 updateRelationTypeWeight()

Input Parameter	Explanation	Optional
RelationType	Designate which RelationType have to modify.	No
KnowledgeBaseDescriptor	Define knowledge base.	
Return Value		
void		

Update the relation type weight. This method update the transitive path to.

3.2.9 UpdateRelationWeight()

Input Parameter	Explanation	Optional
Relation	The type of the relation weight will be updated.	No
weight	The new weight.	No
KnowledgeBase	Define the knowledge base.	No
Return Value		
void		

This method update the transitive path to.

3.3 FullTextSearchInterfaceImpl

The base implementation should use Lucene for FTS. This simplifies all desired methods

since Lucene already has implementations in place for almost all of our desired methods. All fields should be indexed both normal and reversed to allow for PostFixQuery implementing by simply firing a PrefixQuery against the reversed field. Should support indexing “plain text files” HTML files and XML files if the field is defined as an Xpath.

Should have the following fields

title – Title of the document (Stored, Indexed, Tokenized)

content – Title of the document (Indexed, Tokenized)

URI – The URI of the document (Indexed, Stored, Non-Tokenized (i.e. Keyword))

lastModified – The timestamp of last modification (Indexed, Non-Tokenized)

filetype – The filetype of the Document.

3.3.1 addDocument()

Input parameter	Explanation	Optional
String uri	The location to add. If the location is an HTML page crawl the page and follow the links to further index documents until the recursive depth limit is reached. If the location is a folder, crawl the folder recursively indexing all compatible files in the folders and sub folders for up to the depth defined by the depth limit.	No
Int depth	Defines the depth of the recursive crawling or folder crawling that should be performed.	No
Boolean reindex	If true – overwrites old index folder If false – appends the new Documents found during indexing to the already exisiting index	No
Return value		
void		

3.3.2 deleteDocument()

Should allow to delete a document based either on the URI of the document, or the Lucene DocId of the document.

Input parameters	Explanation	Optional
String URI	The URI of the document to delete	No, either this or docId has to be set.
Long docId	The lucene docId of the document to delete	No either this or URI has to be set.
Return value		
void		

3.3.3 doSearch()

Input parameter	Explanation	Optional
SearchObject search	The object represents a search request. It contain which lucene field want search, topic from knowledgebase and the searching value.	No
SearchConfig config 1. boolean fuzzySearch 2. int maxNumberOfHits 3. double resultThreshold	Configure the search engine with 1. True only if we want the query against the lucene to be done as a FuzzyQuery 2. Set the max number of the hits. 3. At which weight above interest the result	No.
Return Value		
A Lucene hits object	For now, for big datasets should implement the HitCollator to allow paging through hits and not having to read through more than a 100 documents at the time.	

Perform a search against the Lucene index, performing necessary parsing of the string passed in to perform the correct type of Lucene query. Should support all Lucene QueryTypes but also PostfixQuery (This is done by indexing the content both front to back and back to front and then performing a PrefixQuery on the reverse index field of the index). If a SearchObject contains Topics, build the query from them too.

3.4 CompassInterfaceImpl

3.4.1 search()

Input parameters	Explanation	Optional
SearchObject	Define the search terms and fields	No
SearchConfig	Define searching options	No
List<KnowledgeBaseDescriptor>	Define which knowledge bases like to use.	No
Return value		
ResultContainer result		

This method first fill up the search config object's members with them default value's if doesn't set. After that stem all words in the search object. In the third step get topic extensions form the knowledge base store Error: Reference source not found and Error: Reference source not found. In the last step call the FullTextSearchInterface's 2.2.1 doSearch(FulltextQuery, FullTextTopicQuery) method and return its result.

3.4.2 getDefaultSearchFields()

Input parameters	Explanation	Optional
Return value		
List<SearchField>	Return with the default search	

	fields	

3.4.3 getAvailableSearchFields()

Input parameters	Explanation	Optional
Return value		
List<SearchField>	Return with all available search fields	

3.4.4 listKnowledgeBases()

Input parameters	Explanation	Optional
Return value		
List<KnowledgeBaseDescriptor>	Return with all stored knowledge bases.	

3.4.5 getAvailableUploadTypes()

Input parameters	Explanation	Optional
Return value		
Set<String>	Return with all available upload types.	

3.4.6 uploadKnowledgeBase()

Input parameters	Explanation	Optional
File file	The file which we want to import.	No
String type	Define the file type	No
KnowledgeBaseDescriptor	Define the knowledge base name	No

	and description.	
Return value		
boolean		

3.4.7 getImportedScopes()

Input parameters	Explanation	Optional
Return value		
Set<Scope>	Return with the imported knowledge base's scopes.	

3.4.8 startImportKBProcess()

Input parameters	Explanation	Optional
Set<Scope>	The set of the scopes which we want to save.	No
Return value		
void		

3.4.9 getRelationTypesInImportedKnowledgeBase()

Input parameters	Explanation	Optional
Return value		
Set<RelationType>	Return with relation types in the imported file and if the knowledge base exist in the store then the set contains the old relation types to.	

3.4.10 storeImportedKnowledgeBase()

Input parameters	Explanation	Optional
Set<RelationType>	The knowledge base will be save with these relation types.	No
Return value		
void		

3.4.11 getRelationTypes()

Input parameters	Explanation	Optional
KnowledgeBaseDescriptor	Define the knowledge base.	No
Return value		
Set<RelationType>	Return with the relation types in the stored knowledge base.	

3.4.12 updateRelationTypes()

Input parameters	Explanation	Optional
Set<RelationType>	The modified relation types.	No
KnowledgeBaseDescriptor	Define the knowledge base.	No
Return value		
void		

3.5 LanguageToolsInterfaceImpl

3.5.1 getStem(s)()

Input parameter	Explanation	Optional
String word	The word to be stemmed	No
Return value		
String stemmedWord	The stemmed word	

For now, should just use Lucene's contrib module Snowball stemmer. Stems the word(s) passed in and returns the stemmed word.

3.5.2 getSpellingSuggestion()

Input parameter	Explanation	Optional
String usersearch	The search to get suggestion for	No
Return value		
String suggestion	The suggested word	

Using the Lucene content index as the base for spelling suggestions, We'll send along an example on how to perform this. Should accept a word and return alternatives from the index. For instance, if the search made was 'isntallation', using the spell suggester could return 'installation' as an alternative; provided the word installation is in the content. Returns the highest scored alternative.

3.5.3 getSpellingSuggestions()

Input parameter	Explanation	Optional
String usersearch	The search to get suggestions for	No
Return value		
List<String> suggestions	The suggested words	

Returns all spelling alternatives instead of just the highest scored alternative. Besides that, works in exactly the same way as 3.5.2 getSpellingSuggestion()

3.6 ImportManagerInterfaceImpl

3.6.1 listAvailableImportPlugins()

Input parameters	Explanation	Optional
Return value		
Set<String>	Return with all available import types.	

3.6.2 importKB()

Input parameters	Explanation	Optional
InputStream	The import stream.	No
String type	Define the import type.	No
KnowledgeBaseDescriptor	Define the knowledge base.	No
Return value		
boolean		

3.6.3 importKB()

Input parameters	Explanation	Optional
File	The import File.	No
String type	Define the import type.	No
KnowledgeBaseDescriptor	Define the knowledge base.	No
Return value		
boolean		

3.6.4 importKB()

Input parameters	Explanation	Optional
URL	The import url.	No
String type	Define the import type.	No
KnowledgeBaseDescriptor	Define the knowledge base.	No
Return value		
boolean		

3.6.5 getScopes()

Input parameters	Explanation	Optional
Return value		
Set<Scopes>	Return with the imported scopes.	

3.6.6 processKB()

Input parameters	Explanation	Optional
Set<Scope>	The set of the scopes which we want to save.	No
Return value		
void		

3.6.7 getRelationTypes()

Input parameters	Explanation	Optional
Return value		
Set<RelationType>	Return with the relation type which are in the imported	

	knowledge base.	

3.6.8 getKnowledgeBaseModel()

Input parameters	Explanation	Optional
Return value		
ImportResult	Return with the import result. The object contains the import warnings and the knowledge base model..	

3.6.9 getKnowledgeBaseDescriptor()

Input parameters	Explanation	Optional
Return value		
KnowledgeBaseDescriptor	Return with the knowledge base descriptor object.	

3.6.10 calculateOccurrence()

Input parameters	Explanation	Optional
Set<RelationType>	Calculate the occurrence for these relation types.	
Return value		
void		

3.7 ImportManagerPluginInterface

3.7.1 importKB()

Input parameters	Explanation	Optional
KnowledgeBaseDescriptor	Define the knowledge base.	
InputStream	The input stream from which the knowledge base definition can be read	
Return value		
InportResult	Return contain the import warnings and the read knowledge base model.	

3.7.2 importKB()

Input parameters	Explanation	Optional
KnowledgeBaseDescriptor	Define the knowledge base.	
File	The file from which the knowledge base definition can be read	
Return value		
InportResult	Return contain the import warnings and the read knowledge base model.	

3.7.3 importKB()

Input parameters	Explanation	Optional
KnowledgeBaseDescriptor	Define the knowledge base.	
URL	The url from which the knowledge base definition can be read	
Return value		
InportResult	Return contain the import warnings and the read knowledge base model.	

4. Demo Web Application

We need a small demo application that demonstrates the use of the Compass API (defined in chapter 2, and a small implementation in chapter 3). It should provide a search box for searches and a result page that both presents the results and gives a good presentation of a the knowledge base expansion. It should also allow the user to play with the preferences. That is adjusting hopcount, threshold, stemming / no stemming. Content for the demo application to use as index will come later.

4.1 Search box

Should accept any search. If the user types prepends their search with 'lucene:' should pass the search directly to the Lucene QueryParser to build the search.

4.2 Configuration toggling

Should allow toggling of all important configuration variables. i.e.

- hopCount – How many hops should the KnowledgeBase performing
- thresholdWeight – What should be the threshold for including an expansion.
- UseStemming - Whether or not the search should be stemmed and used against a (non-)stemmed.
- PrefixMatching – Whether or not the KnowledgeBase should perform prefix matching on topics. i.e. if the user enters 'Know' as the search, should perform a 'Know%' query.
- HitThreshold – If the number of results from the lucene search is less than this, should use the LanguageTools to get spelling suggestions.

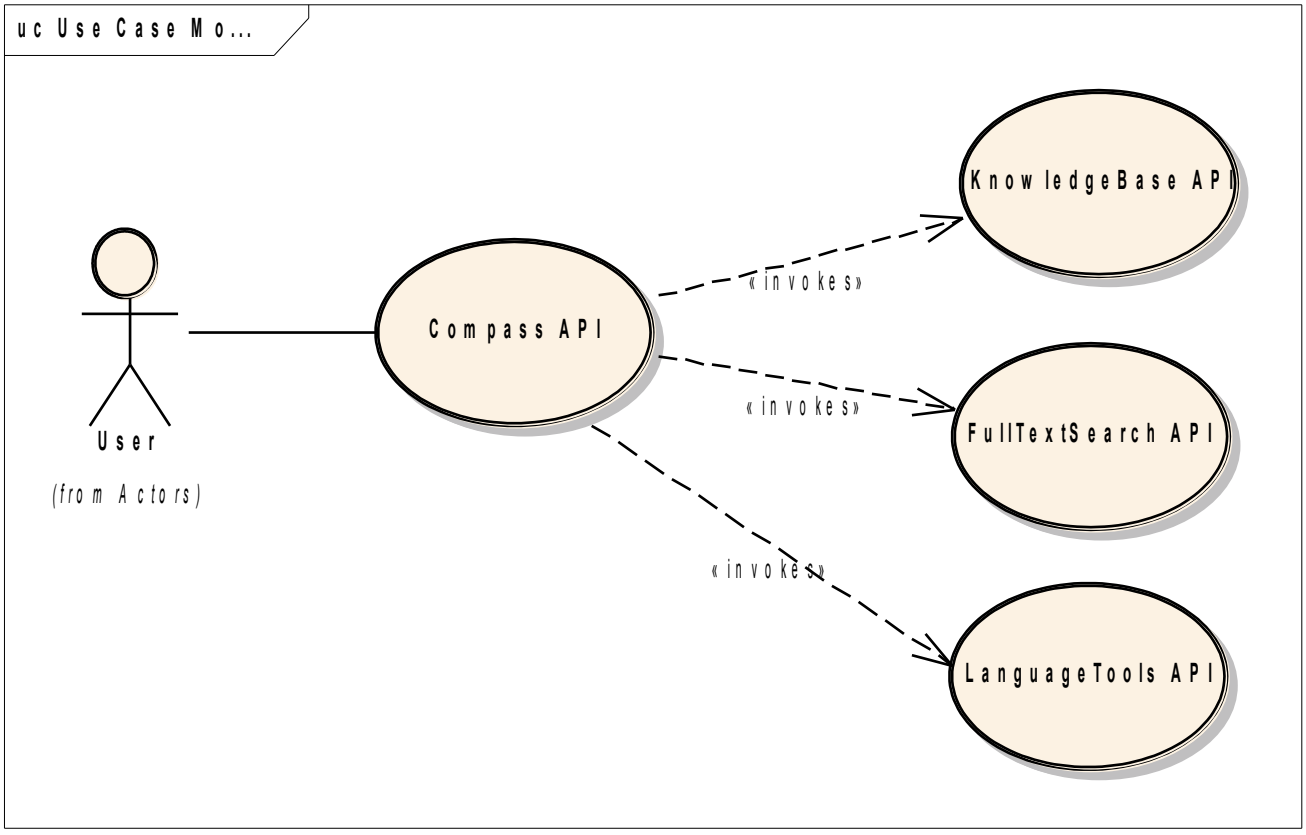
4.3 Result presentation

Presents the results in a sorted list on score from Lucene, but should also have the possibility for sorting hits on the various fields of the Document.

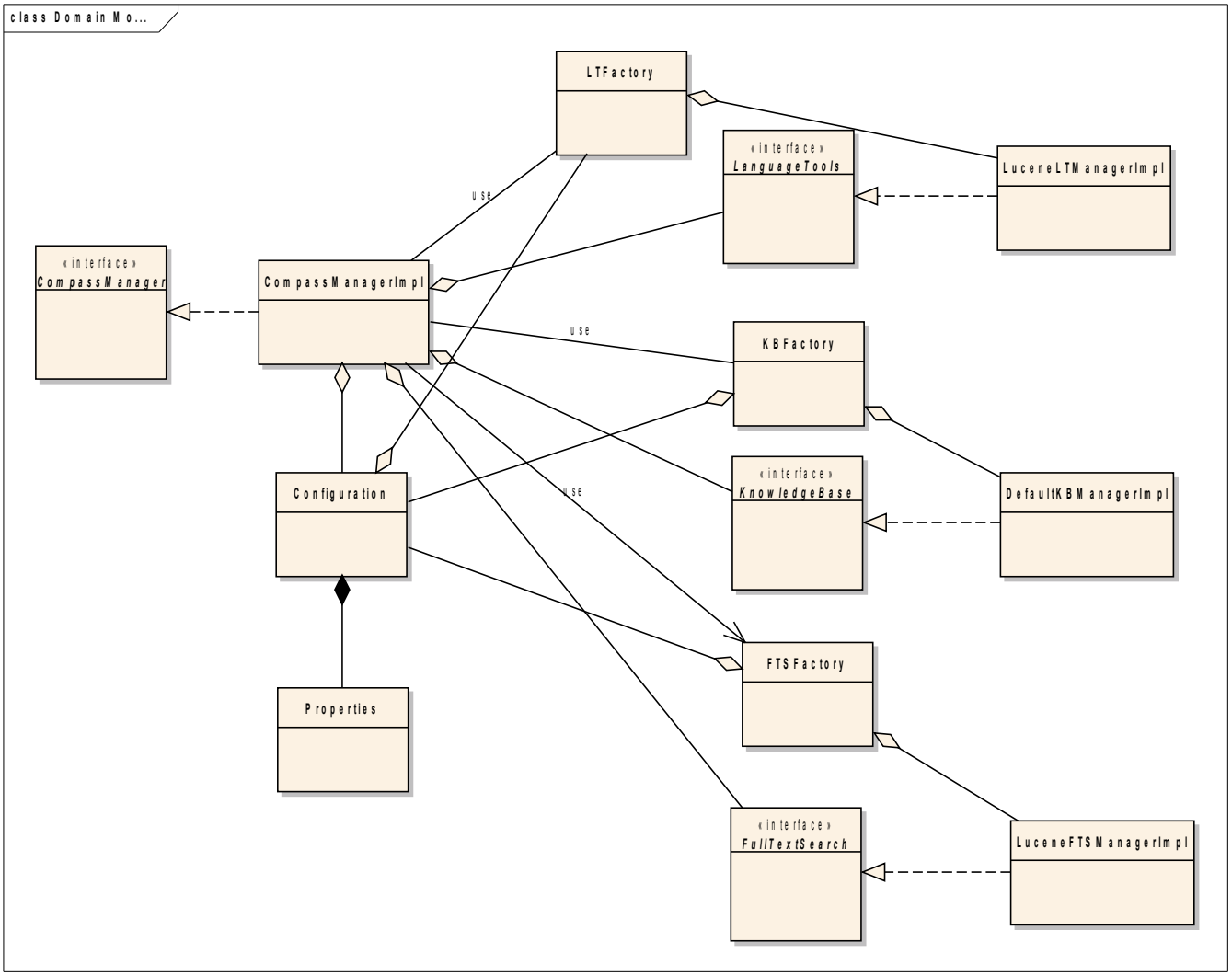
Also presents the topic map expansion in an easy to grasp way, for instance as an expandable tree which shows the user search as the top node and then expands into the expanded topics with as many levels as there was hops, unless no hits above the threshold was found in the later hops.

5. Models

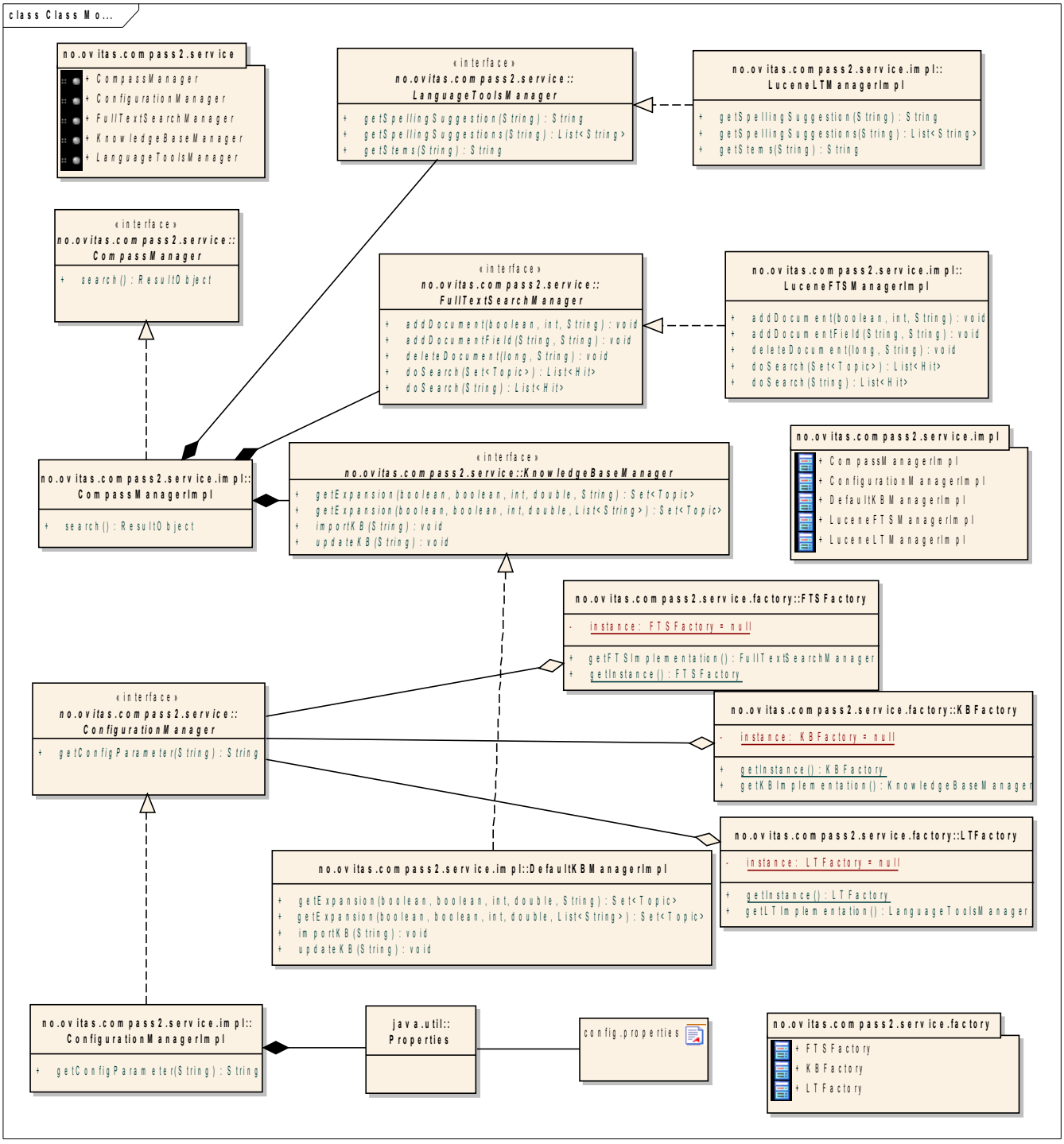
5.1 Use Case Model



5.2 Domain Model



5.3 Class Model



6. Configuration

For the configuration the ConfigurationManager will be used, which is now using XML file.

The configuration file can be divided into five parts. Let's see them.

The xml root element is : `<compass2-config>`

6.1 Full text search part

In this part of the configuration file can set the system how use the full text search service. This settings delegate to object which implements the 2.2 FullTextSearch - interface.

Here is an example:

```
<full-text-search prefix-match="yes" fuzzy-match="no">
  <full-text-search-implementation class="FTSManagerImpl" >
    <params>
      <param name="indexdirectory-path" value="c:/home/lucene/ftsindex"/>
      <param name="document-repository" value="${document.repository}"/>
      <document-fields>
        <field name="title" indexed="analyzed" stored="yes" type="string"/>
        <field name="date" indexed="analyzed_no_norms" stored="no" type="date"/>
      </document-fields>
    </params>
  </full-text-search-implementation>
  <indexer-implementation class="LuceneIndexerImpl" />
  <content-handler-implementation class="TikaFTSContentHandlerImpl" />
</full-text-search>
```

6.1.1 Full-text-search tag

Store all full text search specific configurations.

Attribute parameter	Explanation	Optional
prefix-match (yes | no)	Can use prefix match the implementation.	No.
fuzzy-match (yes | no)	Can use fuzzy mathc the implementation.	No.
Elements		
full-text-search-implementation	Describe the full text search	No

	implementation's configuration.	
indexer-implementation	Describe the indexer implementation's configuration.	No.
content-handler-implementation	Describe the content handler implementation's configuration.	No

6.1.2 Full-text-search-implementation tag

Configure the full text search manager object and store indexer and content handler configurations too because these settings use all of them.

Attribute parameter	Explanation	Optional
class	Set the FullTextSearchManager implementation class.	No.
Elements		
Params?	Store the configurations.	Yes

6.1.3 Params tag

Store the full text search's configurations.

Attribute parameter	Explanation	Optional
Elements		
Param+	Store the generally configurations.	No
Document-fields?	Store which content fields will be stored and indexed.	Yes, then index and store all content field.

6.1.4 Param tag

A param tag is like a property tag in a properties file. It is a name-value pair. In 6.1.2 Full-text-search-implementation tag two recommended parameters' name are:

- indexdirectory-path: which describe where I the indexer base directory.

- Document-repository:

Attribute parameter	Explanation	Optional
name	The parameter name.	No
value	The parameter value.	No.
Elements		

6.1.5 Document-fields tag

Store which content fields will be stored and indexed.

Attribute parameter	Explanation	Optional
Elements		
Field*	Represent a content field define how it will be indexed.	Yes, then index and store all content field.

6.1.6 Fields tag

Define a store, indexing mod and type of the fields.

Attribute parameter	Explanation	Optional
name	Define the content field.	No.
indexed (no | analyzed | not_analyzed | not_analyzed_no_norms | analyzed_no_norms)	Configure the indexing.	No.
stored (yes | no | compress)	Define the storing mode.	No
type (string|number|date)	Define the field type.	No.
Elements		

6.1.7 Indexer-implementation tag

Define the indexer class.

Attribute parameter	Explanation	Optional
class	Set the indexer.	No.
Elements		

6.1.8 Content-handler-implementation tag

Define the content handler class.

Attribute parameter	Explanation	Optional
class	Set the content handler.	No.
Elements		

6.2 Language tools part

6.3 Knowledge bases part

In this part of the XML file is describe the knowledge base service who will be work. The system can initialize the knowledge base object two mode. Te first mode is with reflection as 6.1 Full text search part or using spring.

Here is an example:

```
<knowledge-bases>
  <knowledge-base name="default" spring-mode="true">
    <knowledge-base-implementation
        class="${no.ovitas.compass2.service.KBImplementation}" >
      <params>
        <param name="file-path" value="${knowledge.base.file}"/>
        <param name="load-on-startup"
            value="${knowladge.base.load.on.startup}"/>
      </params>
    </knowledge-base-implementation>
    <knowledge-spring-implementation bean="kbBean" >
      <context-files>
        <context-file file="classpath*:/applicationContext.xml" />
        <context-file file="classpath*:/applicationContext-dao.xml" />
        <context-file file="classpath*:/applicationContext-service.xml" />
      </context-files>
      <params>
        <param name="file-path" value="${knowledge.base.file}"/>
```

```
            <param name="load-on-startup"
                value="${knowladge.base.load.on.startup}"/>
          </params>
        </knowledge-spring-implementation>
        <expansion use-random-weight="no" expansion-threshold="0.7"
            max-nr-of-topic-to-expand="50" hop-count="1">
          <association-types>
            <association-type id="#t-11372557" name="TestAssocTypeName1"
                weight-ahead="1.0" weight-aback="0.1" />
            <association-type id="#t-11372561" name="TestAssocTypeName2"
                weight-ahead="1.0" weight-aback="0.1" />
          </association-types>
        </expansion>
      </knowledge-base>
    </knowledge-bases>
```

6.3.1 Knowledge-bases tag

This tag store the each knowledge base configuration.

Attribute parameter	Explanation	Optional
Elements		
knowledge-base*	One knowledge base configuration.	Yes.

6.3.2 Knowledge-base tag

Configure a knowledge base. Define which method can be built the KnowledgeBaseManager object.

Attribute parameter	Explanation	Optional
Name	The knowledge base name	No
spring-mode (true|false)	If it true then the factory use spring to create instance from KnowledgeBaseManager	No
Elements		
knowledge-base-implementation	If spring mode false use this configuration.	Yes.

knowledge-spring-implementation	If spring mide true use this configuration	Yes
Expression		

6.3.3 Knowledge-base-implementation tag

Define the configuration the knowledge base object.

Attribute parameter	Explanation	Optional
class	Class which implements KnowledgeBaseManager interface	No
Elements		
Params?	Store parameters.	Yes.

6.3.4 Params tag

See in the 6.1.3 Params tag section.

6.3.5 Param tag

See the attributes, elements and how to use in 6.1.4 Param tag. In the 6.1.2 Full-text-search-implementation tag two important parameters need to add.

- file-path:
- load-on-startup:

6.3.6 Knowledge-spring-implementation tag

Define the configuration the knowledge base object with spring usage.

Attribute parameter	Explanation	Optional
bean	The bean name which implement	No

	the KnowledgeBaseManager interface	
Elements		
context-files	Store spring configuration files path.	No.
Params?	Store parameters.	Yes.

6.3.7 Context-files tag

This tag contains context-file tags which define the spring configuration files path.

Attribute parameter	Explanation	Optional
Elements		
context-file+	Store spring configuration file path.	No.

6.3.8 Context-file tag

Define spring configuration file path. The path need path you must to add a correct format or the ClassPathXmlApplicationContext couldn't find the spring bean.

Attribute parameter	Explanation	Optional
file	Define spring config file path in ClassPathXmlApplicationContext format.	No.
Elements		

6.3.9 Params tag

See it 6.3.4 Params tag.

6.4 Search bases part

This part of the configuration file you can define the searching fields, default searching fields.

Here is an example:

```
<search-bases>
  <search-field-string id="content" index-field="content"
      search-field="Content" type="string" default="true"
      weight="1" match="match_all" fit="prefix"/>
  <search-field-string id="title" index-field="title"
      search-field="Title" type="string" default="true"
      weight="0.5" match="match_any" fit="whole_word"/>
  <search-field-dateinterval id="lastModified" index-field="lastModified"
      search-field="Last Modified" type="dateinterval" default="false" />
</search-bases>
```

6.4.1 Search-bases tag

Contain the search fields definition. Important! If not add fields then you can not search!

Attribute parameter	Explanation	Optional
Elements		
search-field-string*	Defines string search field.	Yes
search-field-dateinterval*	Defines date interval search field.	Yes
search-field-numeric*	Defines numeric search field.	Yes.

6.4.2. Search-field-string tag

Define a string search field and add default values for fast search. Generally only search-field-string used for fast search.

Attribute parameter	Explanation	Optional
id	Search field id.	No.
index-field	The search field id in the indexer.	No.
search-field	The search field label in the web plugin.	No.
type(string)	Search field type.	No.
default (true | false)	If true, when perform fast search then use this field too.	No.

weigh	If default true, when perform fast search then use field with this weight.	No.
Match (match_all | match_any)	If default true, when perform fast search and the search value has more terms then use this relation between them.	No.
Fit (prefix | whole_word)	If default true, when perform fast search, it define what type of query will be created.	No.
Elements		

6.4.3 Search-field-dateinterval tag

Define a date interval search field.

Attribute parameter	Explanation	Optional
id	Search field id.	No.
index-field	The search field id in the indexer.	No.
search-field	The search field label in the web plugin.	No.
type(dateinterval)	Search field type.	No.
default (true | false)	If true, when perform fast search then use this field too. Generally false.	No.
Elements		

6.4.4 Search-field-numeric tag

Define a numeric search field.

Attribute parameter	Explanation	Optional
id	Search field id.	No.

index-field	The search field id in the indexer.	No.
search-field	The search field label in the web plugin.	No.
type(integer | float)	Search field type.	No.
default (true | false)	If true, when perform fast search then use this field too. Generally false.	No.
Elements		

6.5 Result part

This part of the configuration XML defines global default configurations. If the SearchConfig object either member not set then the Compass2Manager will add these defaults to the configuration object.

Here is an example:

```
<result result-threshold="0.1" max-number-of-hits="100" hop-count="1"
    prefix-match="false" fuzzy-match="false"
    max-nr-of-topic-to-expand="50" expansion-threshold="0.7"/>
```

6.5.1 Result tag

Attribute parameter	Explanation	Optional
result-threshold	Default result-threshold value.	No.
max-number-of-hits	Default max-number-of-hits.	No.
hop-count	Default hop-count.	No.
Prefix-match (true | false)	Default prefix-match.	No.
Fuzzy-match (true | false)	Default fuzzy-match.	No.
max-nr-of-topic-to-expand	Default max-number-of-topic-to-expand.	No.
expansion-threshold	Default expansion-threshold.	No.
Elements		