ECP 2005 EDU 038098

MACE

Joint Deliverable "Functional prototype for metadata tools and concepts"

Deliverable number	D3.2, D4.3, D5.2, D6.3 (joint deliverable)
Dissemination level	Public
Delivery date	30 November 2007
Status	Final
Author(s)	FIT, KUL, OUNL, FHP

*e*Content***plus***

This project is funded under the *e*Content*plus* programme[1],
a multi-annual Community programme to make digital content in Europe more accessible, usable and exploitable

[1] OJ L 79, 24.3.2005, p. 1.

Table of Contents

Purpose of this deliverable

This deliverable describes concepts and functional prototypes developed in MACE. Its goal is to describe the prototypes for metadata enrichment developed in the MACE project so far.

As a joint deliverable, it is a collection of the following deliverables listed in the Description of Work:

- D3.2 Functional Prototype for usage metadata
- D4.3 Functional Prototype for contextual metadata
- D5.2 Functional Prototype for competence and process metadata
- D6.3 Functional Prototype for content and domain metadata

For each deliverable, a separate chapter is included so that references to the planned deliverables can be derived easily.

In addition, this deliverable is strongly connected to Joint Deliverable JD5: "MACE toolset and infrastructure, prototype", also due in M15.

Introduction

The project set out to enrich digital contents with metadata. In the project description, we decided to use four specific types of metadata. These types are 1) content and domain metadata, 2) usage metadata, 3) competence metadata and 4) contextual metadata. Put together, they cover a wide range of areas and provide a basis for further interesting tools and solutions.

The work is based on previous steps done in MACE, as were requirements analysis and several fruitful discussions among project partners to understand the scope of the project domain, which resulted in the creation of a joint deliverable titled "Metadata taxonomy and their integration in MACE" in month M9. Also developed were first versions of the “Analysis framework” (D2.1.1), the “Validation framework” (D2.2.1) and an accompanying “Quality control plan” (D2.3.1).

Based on this work as well as work package internal discussions and proposals, we were able to develop prototypes for each of the different types of metadata. The development done so far, including the technical infrastructure to setup and maintain gathering and storage of metadata is described in the next chapters of this deliverable, followed by a conclusion and an outlook.

Functional Prototype for usage metadata (WP3)

Objective

The prototype enables the capturing and processing of events that occur within the MACE infrastructure. Such events are called usage metadata; as such data describes the usage of learning objects within MACE.

Usage related metadata include

- Attention metadata that capture what users actually do with learning objects by way of evaluating log files,
- Annotations that capture explicit feedback from users, including Blog and Wiki comments,
- Folksonomy metadata that capture simple search terms users deploy to describe content that they publish or that they are looking for, and
- Social recommendations that users make to explicitly share content with peers, learners (friend of a friend – FOAF[1] or the "related items" principle from Amazon).

The prototype will enable the usage related metadata functionalities within the MACE infrastructure such as recommendations, guidance on appropriate use, popularity, etc.

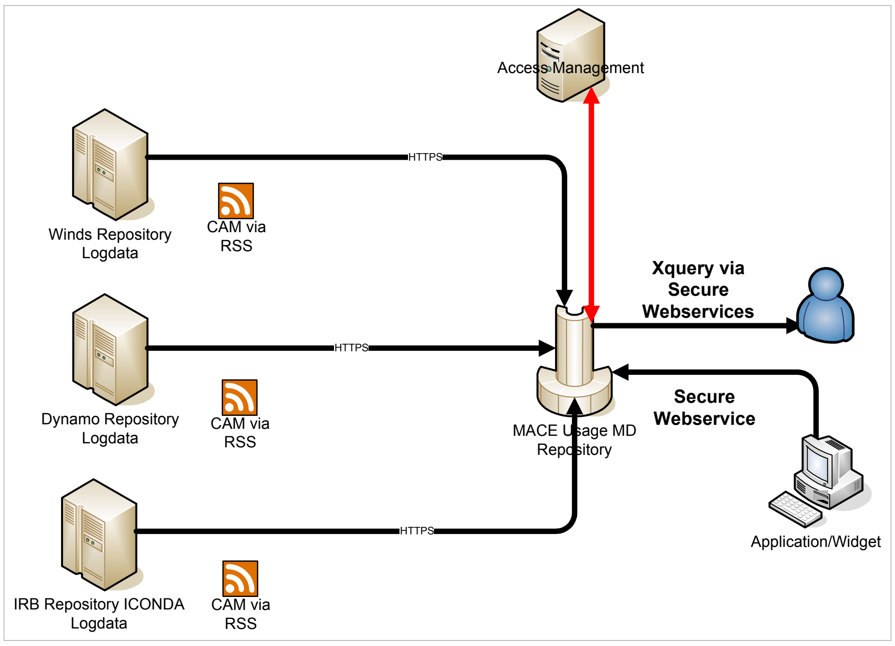


Figure 1 - Usage metadata infrastructure

[1] http://en.wikipedia.org/wiki/Friend_of_a_friend

Prototype Description

Usage metadata sources provide access to the user activities within the MACE infrastructure. Two types of sources exist: content repositories that provide learning objects and the MACE user interfaces. A simple hook into the MACE middle layer provides access to all activities within the MACE system. For redundancy and evaluation purposes, we also capture the usage metadata directly from the repositories.

Based on the collected usage data, we enable ranking and statistics services that provide an improved access to suitable learning objects through an appropriate ranking showed on the user interfaces.

The usage metadata infrastructure is based on the concise application of the Contextualized Attention Metadata (CAM) Schema. This schema describes activities in relation to the learning objects and the users, thus allowing us to enable advanced (personalized) ranking metrics.

All user activities are captured in the MACE infrastructure, expressed as CAM instances, and stored in the central CAM store using the Simple Publishing Interface (SPI.) This push technology enables the MACE system to take all, even the most recent, activities into account when providing ranking metrics for personalized search results.

The content repositories provide their usage data through a RSS stream, one per repository. The RSS streams are read once every 24h from the usage metadata capture engine via secured http connections and stored in the CAM store. The schedule is setup so that the recent RSS streams are read before they are replaced with streams with the new events. The read streams are not deleted but renamed and stored for backup purposes.

The ranking metrics are provided through web services. Depending on the nature of the ranking metrics, they are either pre-calculated in specific indices to enable fast responses, or are calculated at runtime to take the most recent user events into account. The ranking services provide three metrics at the moment:

- Number of downloads
- Number of downloads per user
- Timeline of usage per learning object

Technical description

The prototype enables capturing the user activities from different environments in order to provide new functionality, such as recommendation, guidance on appropriate use, popularity ranking, etc. The prototype is based on two major components (Figure 1): the usage metadata repository and the usage metadata services.

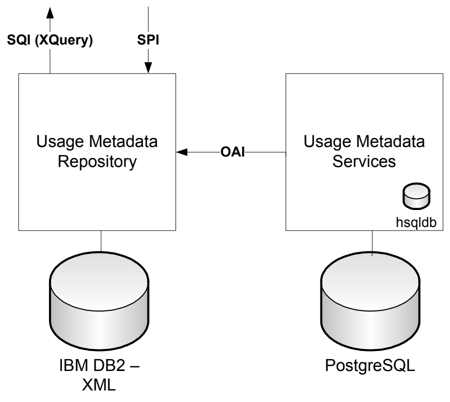


Figure 2 - Usage metadata architecture

Usage Metadata Repository

Based upon the AriadneNext Metadata Store, this central repository stores, from different sources, all the user activities described in the Contextual Metadata Schema (CAM[1]). The repository provides 3 services:

- Simple Publishing Interface (SPI[2]), used to insert XML instances in the repository.
- Simple Query Language (SQI[3]), used to query the repository using the ProLearn Query Language (PLQL[4]).
- Open Archives Initiative Protocol (OAI-PMH[5]), used to expose the CAM instances to a harvester.

The Metadata Store uses XML-enabled database IBM DB2. The inserted XML instances are stored without any pre-processing. Currently, the prototype for usage metadata makes use of the SPI and the OAI services.

[1] http://ariadne.cs.kuleuven.be/hmdb/index.php?option=com_content&task=view&id=28&Itemid=56

[2] http://ariadne.cs.kuleuven.be/lomi/index.php/SimplePublishingInterface

[3] http://ariadne.cs.kuleuven.be/lomi/index.php/LorInteroperability

[4] http://ariadne.cs.kuleuven.be/lomi/index.php/QueryLanguages_v1.0

[5] http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

Providers of usage metadata use the SPI protocol to push CAM instances into de repository. The OAI services are used by the usage metadata services component to obtain the required metadata instances.

Usage Metadata Services

The services component has been developed in order to calculate and provide ranking and statistics services that can be used by any authorized client application through web services. Currently we use Apache Axis 2 as the core engine to create the web services interfaces and the Apache Tomcat 5 as the application server.
Internally the component uses two databases (Figure 1): the normal (non-embedded) database PostgreSQL and the HSQLDB as embedded database to the application.
The non-embedded database duplicates the user activities information in the usage metadata repository and translates them into the relational paradigm. The database is also responsible for the support of the statistic services offered that do not require big calculation times.
The usage metadata component has an internal job scheduling system that manages the update of the non-embedded database. By using OAI Protocol, this component can be configured to automatically harvest new metadata from the repository and insert it into the non-embedded database.
The embedded database is specifically used to enable fast response times to service requests. It stores pre-calculated (complex) ranking metrics supported as service features. The database is pre-populated during the web application loading, by using the non-embedded database to obtain the necessary data to calculate the ranking metrics. All calculations obtained for a single ranking metric are stored in a table of the database; meaning that when one service is required, the component uses this table and quickly responds to the needs of the user application.
All the rankings metrics are calculated again after an automatic harvest has been done to keep the embedded database up to date.

Integration into MACE Infrastructure

Integration into the MACE infrastructure is based on the MACE web service infrastructure. We expect that the ranking metrics are not called directly by the user interfaces. Instead, we expect the MACE middleware to use the metrics service, e.g. for ranking the search result sets. The interfaces can be accessed at this URL:

http://ariadne.cs.kuleuven.be/MetricServiceInterface/services/RankingMetrics?wsdl

Ranking Metrics Description

You can use the *generalGetRankingMetricsDescriptions* function to obtain a list of the available metrics. The response will show the identifier of all the available metrics, a description of the calculation that is performed, and the parameters needed for their calculation.

Function Name:	**generalGetRankingMetricsDescriptions**
Parameters:	**None**
Return type:	**String**

Obtain Ranking Metrics Calculation

To obtain a list of the top-k (k) objects ranked according to the specified metric (metricId) on the last n period (*timePeriod*), you should use the *generalGetRankingMetricValues* function, described below.

Function Name:	**generalGetRankingMetricValues**	
Parameters:	**Name**	**Type**
	k	Integer
	metricId	String
	params	String[]
	timePeriod	Integer

The parameters for the metrics can be found analyzing the result of the call to *generalGetRankingMetricsDescriptions* function, described above. The time period can be specified with the following values:

Value	Meaning
1	1 day
2	1 week
3	1 month
4	1 year
5	Since recorded history

Further plans

The MACE usage metadata infrastructure at present provides basic functionality. It will be extended with additional and advanced metrics for ranking purposes and recommendation functionality. Such extensions need to take the specific nature of the activities of users within the MACE system into account. Therefore, a rather large effort will be necessary to adapt existing solutions to the requirements of MACE.

In addition, usage metadata will be applied to generate a number of statistics, e.g. number of learning objects used, time and date of usage, etc. The statistic information serves as a base for recommendation features like "users who read this also read…"

Functional Prototype for contextual metadata (WP4)

Introduction

Contextual metadata are used to describe a situation or connections between seemingly unrelated contents. In the architectural education domain concerned by the MACE project, they can be used to enable students to learn about connections that were not visible before. We hope to find out that students can learn more effectively when being empowered to search in repositories they did not even know about before.

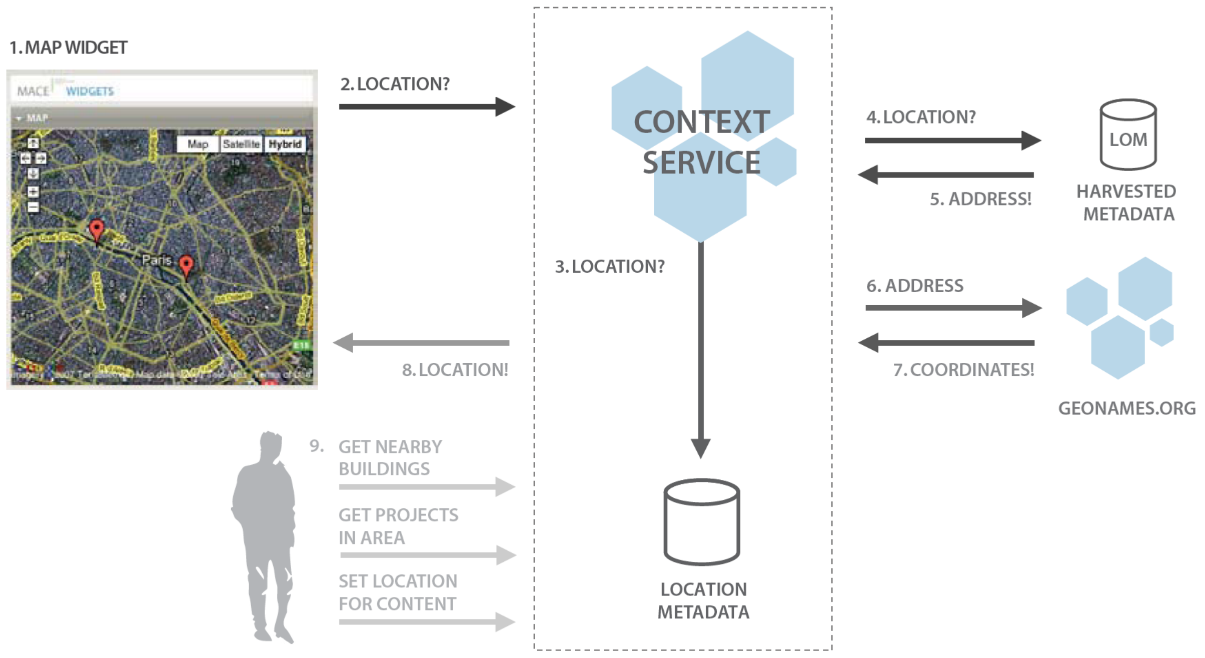


Figure 3 - ContextService prototype (system view)

The prototype for contextual metadata works on a specific problem: relating contents to a location they are describing. For example, a building created by Renzo Piano, the famous architect, is described as being located in Berne, Switzerland. By employing some clever mechanisms to match contents against geo-coordinates, we can pinpoint that content on a map and also show other contents mentioning Berne next to it. These contents do not necessarily be about Renzo Piano, too. In fact they can be anything else, from regulations to travel information, and come from different repositories. We “simply” use the physical location context to connect them to each other. While this does not make sense in all situations, for a great number of cases the use of location context as “glue” between (learning) contents will provide new and interesting insights to the student.

System design questions

For the prototype being able to handle these tasks, some problems needed to be addressed:

1. How to get location information (context metadata) from contents and repositories?
2. How to translate this information into geo-coordinates?
3. How to display these coordinates on a map?
4. How to design user interaction, so users can actively update context information, i.e. create new context metadata?

To solve the first two problems, we created a full text index of all contents and matched that index against GeoNames[1], an online geographical names database. GeoNames consists of over six million locations and also provides alternate spellings for most of them, so searches for “Florence”, “Florenz”, “Florència” and “Firenze” all lead to the same location record. This is very useful because the content we indexed was in at least three different languages (English, Italian and German). GeoNames also has become the basis for several other projects[2], which helps with adding new location information and correcting existing information, thus making GeoNames even more valuable for us.

To create the full text index, we used Lucene[3], a high performance index creation and management utility. The index was created by feeding to Lucene all contents from the WINDS database, a process which took about 30 minutes and resulted in a index with about 48400 non-duplicate entries (so called index-terms), each of them having between one and eleven occurrences in actual documents. For example, the term “Barcelona” occurs in eight WINDS contents. From GeoNames, we took the list of alternate location names (available for download from their site[4]). This list after cleanup (duplicates and ignoring capitalisation) contains 2.232.500 entries.

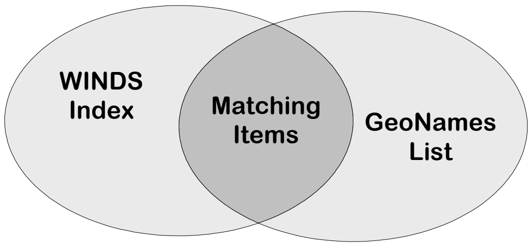


Figure 4 - Matching WINDS and GeoNames

[1] http://geonames.org

[2] http://www.geonames.org/users.html

[3] http://lucene.apache.org/

[4] http://download.geonames.org/export/dump/alternateNames.zip

As a result of the matching operation, we got a collection of about 3600 hits. Several of these hits are not real locations but abbreviations, for example of people names. We needed to find a way to deal with these mistakes, which will be explained in a moment. First, we want to explain how to visualize the results.

To visualize items on a map, we choose the Google Maps API because it is mature, well documented and used in many projects already, which eases development and also increases chances that users might have seen and used similar tools already.

Google Maps provides a highly customisable map that users can interact with in standard, but now also in new and MACE-specific ways. She can select an appropriate section by panning and zooming the map, thus getting the possibility to explore specific areas in more detail, or acquiring an overview by zooming far out. Furthermore, we allowed the user to switch between satellite (satellite and high-resolution aerial photographs) and map view (topographic and street map), to provide different aspects of context to view and compare.

The Google Map API makes it easy to create own simple map applications. We have embedded that map not only in some web page, but integrated it into our widget framework. The *MapWidget*[1] loads contents dynamically via Asynchronous JavaScript and XML (AJAX[2]).

A main content can be displayed, with further related contents to be fetched, enabling the user to not only view the location of the currently chosen content, but also to analyse the environment and interesting contents in the vicinity. Besides this interaction mechanism, the user can pick another area by zooming and panning, so the widget loads all contents to display by passing the coordinates of the selected region.

For every single content on the map, further information can be requested. The user can obtain these by simply clicking on one of the shown markers, so that a small popup window with additional metadata appears. Furthermore, an authorised user may place a content object on the map for the first time, re-place it if its prior location has been erroneous, or refine its metadata in a shortly displayed detailed map, if the location was correct but not exact enough. The new geo-coordinates then are sent to the ContextService for processing and storage.

1 http://interface.fh-potsdam.de/mace/widgets/mapWidget/

2 http://en.wikipedia.org/wiki/Ajax_(programming)

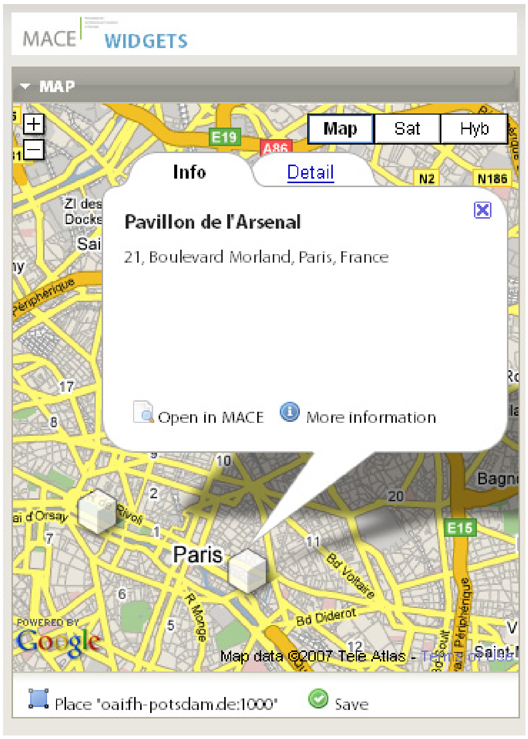


Figure 5 - MapWidget (screenshot)

With the user being able to see contents on a map, mistakes will be recognized quickly. As explained above, a content object can be mistakenly placed or should not be placed at all because it does not refer to a location but to a person's name, for example. In this case, the user is able to correct the placement or to remove the location metadata from the content completely. The user, because of her background knowledge in the respective field, is much better in recognizing these mistakes than any automated process.

Because of this, the ContextService offers methods to correct the location information and to update the metadata associated with the content. While this is not exactly newly generated metadata, it is manually updated and therefore can be ranked with higher credibility than automatically generated context metadata.

Prototype development

The context prototype is developed as an AXIS web service, running in an Apache Tomcat container. It can be accessed via this URL:

http://www.mace-project.eu/mace/services/ContextService?wsdl

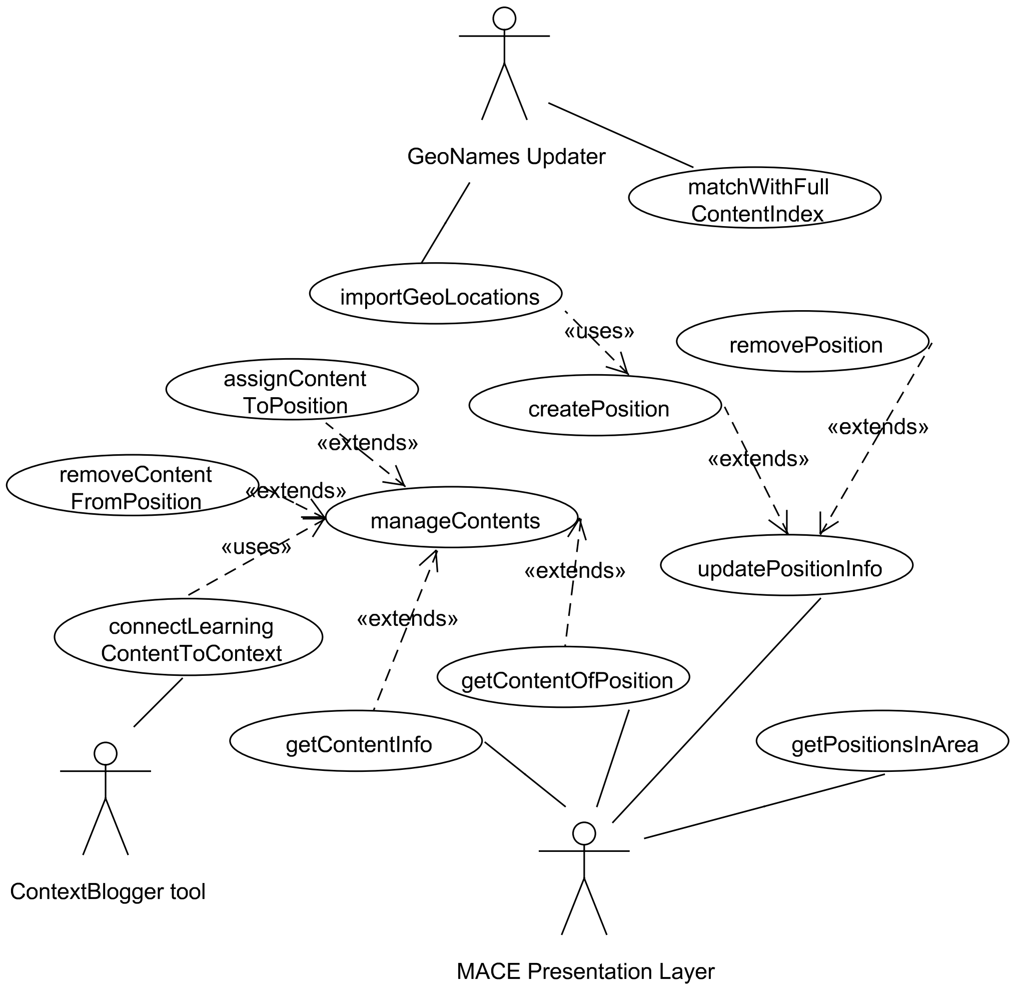


Figure 6 - Use cases for contextual metadata prototype

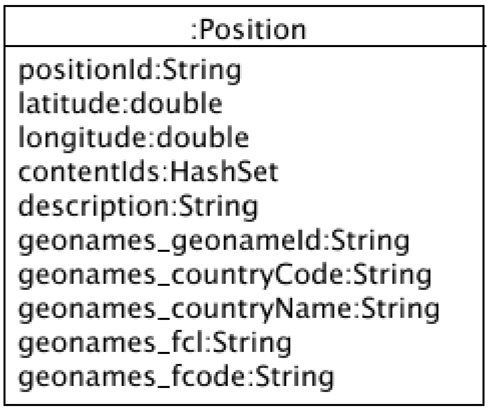
As described above, the prototype version of the ContextService relates digital contents to geographical positions. To achieve this, the service uses its own database. The crucial point is, that the ContextService does not store digital contents in its database, but content identifiers only, which are unique throughout the whole MACE system. The service implementation is based on a three-tier architecture, where the presentation layer is realised by the Map Widget.

Business logic layer

In the prototype version of the ContextService, we used relations of type [1..n] to associate geographic positions to digital contents. This means that a geographical position can be associated to multiple contents, but each content object is related to a single position only.
This makes sense when buildings, cities or other physical objects with a fixed position are considered. In a later version, also persons and other mobile objects will be considered in

MACE, thus we will replace the [1..n] relations by [n..n] relations, so that each object can relate to multiple positions and vice versa.

The [1..n] relations between positions and contents is implemented as follows: For each geographical position, an object of type *'Position'* is created.



```
:Position
positionId:String
latitude:double
longitude:double
contentIds:HashSet
description:String
geonames_geonameId:String
geonames_countryCode:String
geonames_countryName:String
geonames_fcl:String
geonames_fcode:String
```

Figure 7 - Position class overview

The class 'Position' is shown in figure 2. Its fields can be classified into three categories:

1. Geographic information. With each Position object, latitude and longitude values are stored. Additionally, each object has a unique identifier and a textual description.
2. Associated content Ids. These are represented by MACE-wide unique identifiers and stored in a field of type HashSet.
3. GeoNames-related fields. As described above, we use the geonames.org database to associate contents with positions. Geonames.org does not only provide latitude/longitude values, but also additional information with a position. The most important ones are stored by the ContextService, when positions are queried from the geonames.org database. These are *'countryCode'* and *'countryName'*, which store the country (coded and full text), where an object is located, and *'fcode'* and *'fcl'* which store a position's feature code classification. Geonames.org provides a so called 'feature code list' (http://www.geonames.org/export/codes.html), a kind of taxonomy, that classifies positions according to their characteristics (Is a place populated? Is it a city or a village etc?).

The ContextService provides public functions that are used for creating and manipulating positions and their relations to contents. These can be called via the web service interface. Beside these functions, the business logic layer also realises the integration of the geonames.org database. Information from geonames.org is currently 'harvested' and stored statically rather than dynamically linked to the ContextService database for speed reasons. Associating information from geonames.org to our contents is based on a list of terms that occur in our digital contents as well as in the GeoNames database. If a term occurs in a content object, this content is associated to all positions received from the geonames.org database when initiating a full text search with the same term.

Database backend

We use *Hibernate*[1] to persistently store Java objects in a database. The database consists of two tables, one to store positions and one to store content-IDs. The position that a content object is related to is stored in form of its identifier in the content-table.

The web service interface

The ContextService interface provides the below listed, self-explaining methods. Besides the basic types *xsd:string* and *xsd:double* the complex data types *'Position'*, which is explained above, and *'StringArray'* are used for data exchange. The *'StringArray'* type is a workaround to exchange two-dimensional arrays of type *'String'* over the web service interface. It is used by the method *'getContentsOfMultiplePositions.*

- *String assignContentToPosition(String positionId, String contentId)* assigns the content with ID 'contentId' to the Position-object with ID 'positionId.
- *String[] getAllPositions()* lists all Position-objects that are stored in the database.
- *Position[] getPositionsInArea(double nwLat, double nwLon, double seLat, double seLon)* returns all Position-objects that are located in an rectangular area specified by the four parameters of type 'double'
- *Position getPositionOfContent(String contentId)* returns the Position-object that is related to a given content.
- *String removeAllContentsFromPosition(String positionId)* removes all contents from the ContextService's database that are associated to a given Position-object.
- *Position[] getPositionsInRange(double lat, double lon, double range)* is similar to *getPositionsInArea*, the difference is that the area, to which the returned Position-objects belong, is circular and specified by a centre/radius pair.

[1] http://www.hibernate.org/

- *StringArray[] getContentsOfMultiplePositions(Position[] positionIds)* is a wrapper for *getContentsOfPosition*. Multiple requests for *getContentsOfPosition* can be encapsulated in a single request.
- *Position[] getPositionsOfMultipleContents(String[] contentIds)* is a wrapper for *getPositionOfContent* to reduce the number of requests.
- *createPosition(String positionId, double latitude, double longitude, String description, String geonames_geonameId, String geonames_countryCode, String geonames_countryName, String geonames_fcl, geonames_fcode)* creates a Position-object. The GeoNames-related fields are optional.
- *String removeContentFromPosition(String positionId, String contentId)* removes a single content associated to the Position-object with ID 'positionId'.
- *String removePostion(String positionId)* removes a Position-object and all associated contents.
- *String getContentShortInfo(String positionId)* provides a short information about the content with ID 'contentId'. This method is not implemented yet.
- *String[] getContentsOfPosition(String positionId)* returns all contents' Ids that are related to the Position-object with ID 'positionId'.

Future work

Although the only currently planned client will be the widgets also developed in the MACE project, the service is in principle open to anyone interested. By connecting to the above URL with a code generation tool, client interface classes for a lot of programming languages can be created automatically. The ContextService WSDL is WS-I[1] compliant.

Automatically associating contents to geographical context does not always lead to the aspired goal. As already mentioned above, we have to evaluate, how usable the full text matching method is and how it can be adapted to yield better results. A second goal for the future is to include other than geographical contextual information in our metadata.

If the current prototype turns out to work satisfactorily, we will analyse other context settings potentially helpful in architectural education and create additional (web) services to exploit these situations to gather and use contextual metadata in them.

[1] http://en.wikipedia.org/wiki/WS-I

Use of MACE services by ContextBlogger

The ContextBlogger toolset is being developed at OUNL and has already been described in papers[1,2]. It aims to provide mobile and contextualised information access to the MACE services. With the addition of a mobile infrastructure to the already existing services, ContextBlogger provides a way to instantly create and enrich content in a specific real-world context.

Instant access to information with mobile devices is extended by adding information about the user's current situation or context. For example, a picture of a building can be created by using a mobile phone and at the same time enriched with GPS[3] location metadata corresponding to the location the picture was taken. Another way of relating information to a user's physical environment is the use of identification tags attached to physical objects. In the system presented here we use a specific kind of data matrix symbols called semacodes (machine-readable ISO/IEC 16022 data matrix symbols that encode Internet Uniform Resource Locators (URLs). It is primarily aimed at being used with cellular phones, which have built-in cameras)[4]. The available learning content can thus be related to physical objects and in that way directly integrated in a learning process that involves those objects.

The ContextBlogger application combines mobile social software with information about the context of a learner. Based on these underlying concepts and relations several use cases can be developed.

[1] http://dspace.ou.nl/handle/1820/828

[2] http://dspace.ou.nl/handle/1820/1008

[3] http://en.wikipedia.org/wiki/GPS

[4] http://en.wikipedia.org/wiki/Semacode

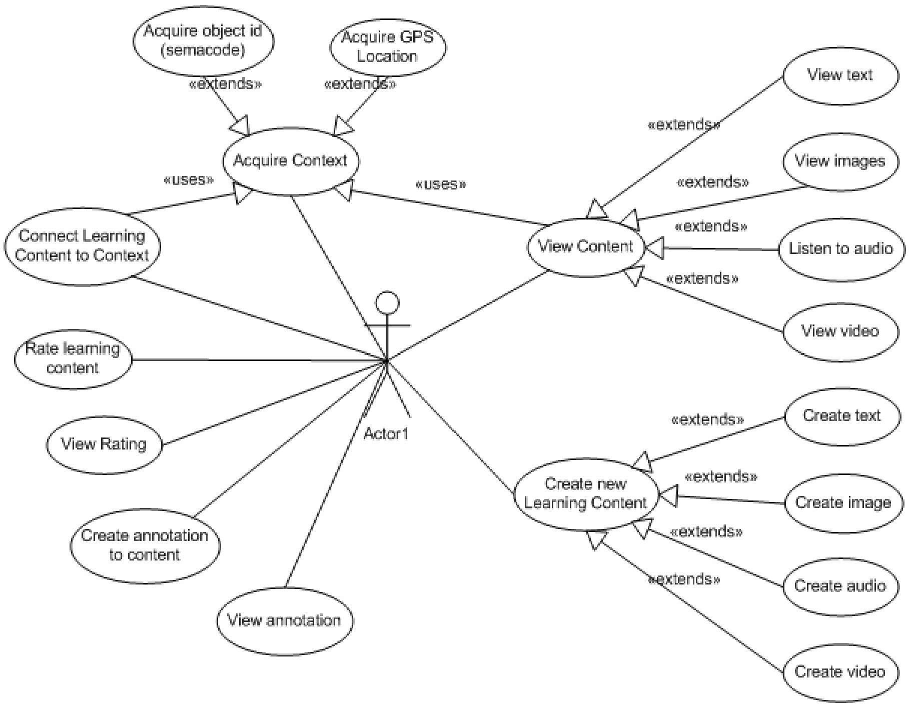


Figure 8 - ContextBlogger use cases

These use cases have led to a technical framework, with the following three subsystems:

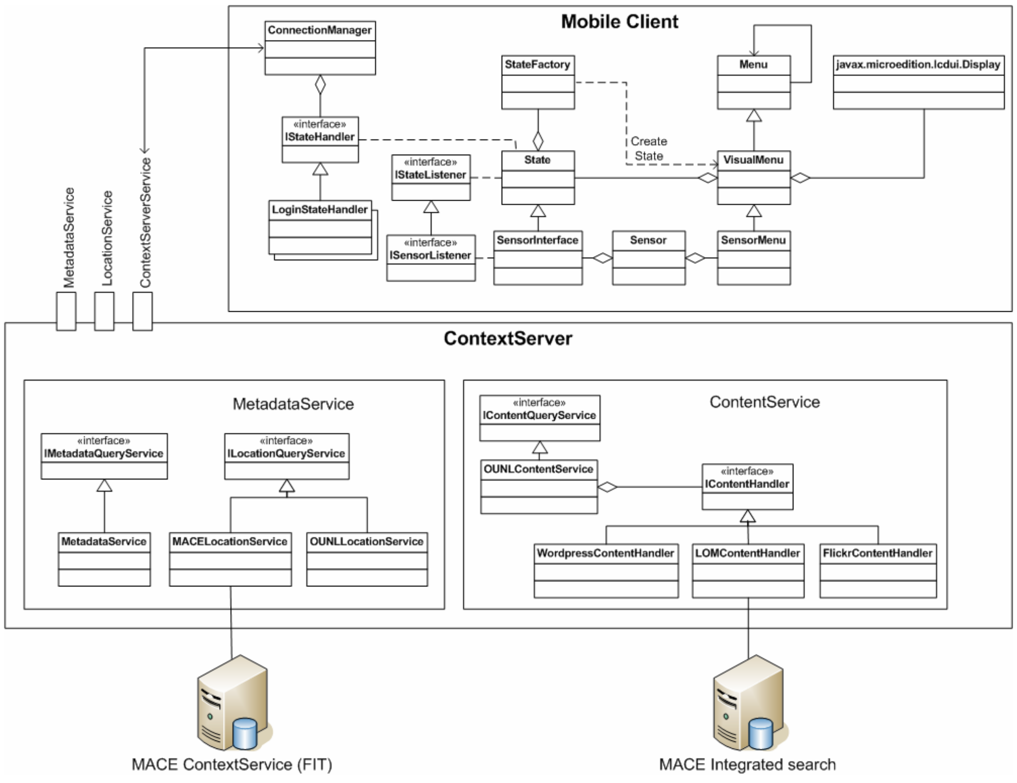


Figure 9 - ContextBlogger infrastructure

- *a mobile client subsystem*, that handles the interaction with the user in the real-world,

- *a content subsystem,* that stores the information that is used to enrich the interaction with the real-world objects,
- *a contextual metadata subsystem* that stores contextual information that relates the context tags with the content in the system.

Available web services

The functionality of the server architecture is made available via a couple of web services, each of which group a certain set of functionalities given by a server module. At this moment, three different web services are available:

1. The *ContextServerService*, which provides the entry point for the mobile client and thus provides all functionality for mobile access. The *ContextServerService* is located at: http://145.20.177.33:8181//ContextServer/ContextServerService?wsdl
2. The metadata service that provides an API for querying all available locally stored metadata. The metadata service is available at: http://145.20.177.33:8181//ContextServer/MetadataServiceService?wsdl
3. The location service, which is a more specific version of the metadata service that enables access to the locally stored location metadata; most of which are GPS locations. The location service can be accessed at the following location: http://145.20.177.33:8181//ContextServer/OUNLLocationServiceService?wsdl

Future work

At the moment of writing, the ContextBlogger software is still in beta stage and not all functionality has been implemented yet. In the future, we will focus on the following:

- Further work on the integration of ContextBlogger with the MACE Context services,
- The integration of the software with the content repositories of MACE, so that content created with the ContextBlogger can be inserted into MACE content repositories with an appropriate LOM description. Moreover, the deliveries of content from within the MACE content repositories.
- Research in combining the map widget and the ContextBlogger for creating ubiquitous learning scenarios.

Functional prototype for competence & process metadata

The rationale behind

Competences and learning processes are often implicitly used in training and education only in the recent years, competences are more and more explicitly used to structure curricula, plan personal development plans and other educational activities. To metatag learning objects with information about competences is a difficult task and requires expert knowledge not only about the domain but often also about the underlying pedagogy. In a first step for the infrastructure for MACE in WP5 the consortium created a flexible set of applications to collect and catalogue competence descriptions, manage and maintain those descriptions and offer an open API to integrate services based on such a competence catalogue into different end user tagging applications.

How the system works

The competence catalogue is a completely object oriented application written in Java and is able to output the data in several output formats such as XML and JSON[1]. A web service layer is developed on top of this application to provide access to the variety of methods in the competence catalogue. This web service is deployed on a Glassfish[2] v2 application server and can be accessed using its SOAP[3] API.

The latest version of the documentation is on the OU.nl MACE Wiki:
http://mace.ou.nl/doku.php?id=competenceserviceapi

[1] http://json.org/json-de.html

[2] https://glassfish.dev.java.net/

[3] http://www.w3.org/TR/SOAP/

System overview

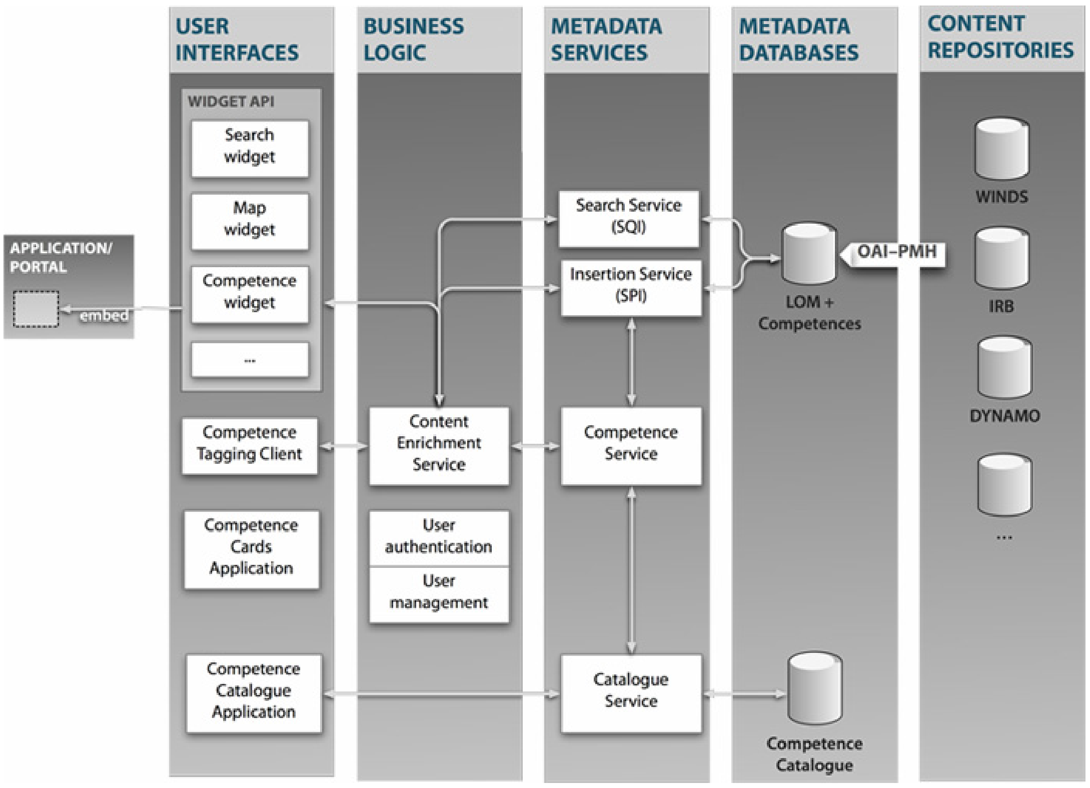


Figure 10 - System overview

Use cases

The MACE Competence Catalogue API makes it possible for a user to search for a competence in different ways. You can search a competence using a full text search on the title/description of the competence or you can search for competences in a specific domain of just get the competence(s) for a certain resource, expert or evidence. If you just want to browse the catalogue you can get a list of all the competences in the catalogue.
If you have found a competence you can start adding metadata to it, such as resources, evidences and experts. The added metadata is weighted so you can see in how much it contributes to that competence. After adding metadata a competence card can be obtained from the web service, which will contain all relevant information combined into an easy to understand XML document.

Metadata generation

The competence catalogue is providing competence metadata and usage metadata. The metadata is generated and stored with the LOM sets, and the catalogue application is used as a reference and vocabulary. The vocabulary service can maintain and manage different domains and competence catalogue for those domains, furthermore it allows to stores relevant resources, experts, and other information with a competence.

Integration with other MACE components and services

As the current prototype backend OUNL has developed a competence administration application. This is a web-based application that supports the management, and administration of different domains and competence descriptions for those domains. Furthermore it offers a service API for manipulation and editing of this information and the use in metatagging.
The competence services are integrated with the MACE Enrichment Widget to quick-tag an object with a competence. Basically user can switch to competence metatagging in the MACE portal and get a tagging field the text in the field is then auto completed according to the vocabulary that has been requested from the vocabulary service. In a next step the MACE Competence Widget will be designed to connect the competence tagging with several other services which will allow adding some granularity to the competences connected to an object such as weighing and relation between competences. The consortium planes to evaluate different approaches for competence tagging beginning 2008. Furthermore an independent client application will be developed that enables competence tagging of arbitrary objects and URIs even outside the MACE learning objects to integrate external references. To implement support for the complex interconnection between the management, maintenance, and description of competences the Competence Card Application is under development. The competence card application gives an overview of a competence and all its related information such as experts, evidences and resources and integrates those information resources in one application.

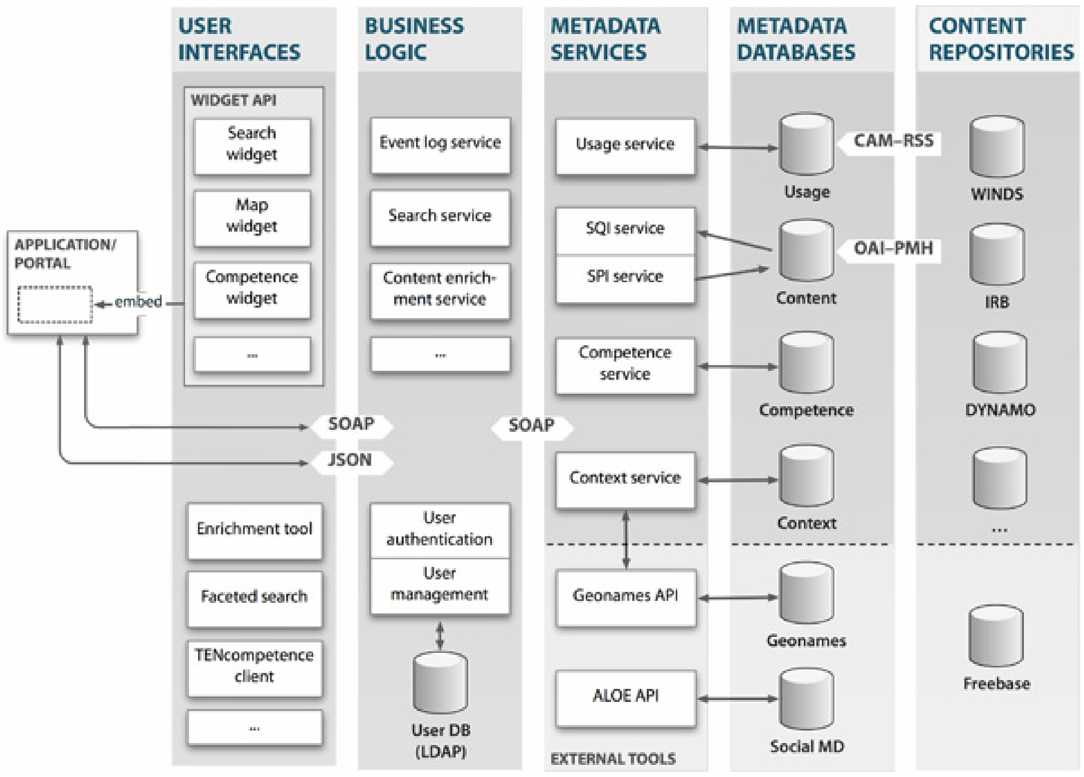


Figure 11 - Infrastructure design overview

A detailed description of the implementation and all APIs can be found at:
http://mace.ou.nl/doku.php?id=competenceconceptions

Furthermore in the next version of the competence catalogue will have:

- full competence maps: support for combining competences in weighted maps and competence profiles which are often associated to job profiles and educational profiles.
- full support for importing competences (HR-XML and RDCEO)
- full support for exporting competences (HR-XML and RDCEO)
- CAM RSS usage metadata interface to track information about usage and social annotation of competences.

Functional Prototype for content and domain metadata (WP6)

Objective

The prototype will deliver a stable infrastructure for metadata and content gathering and access, management and provision within MACE. The infrastructure will enable content enrichment of existing contents with keywords and ontologies from different domains using the MACE application profile.

MACE application profile

The MACE application profile is described in Deliverable 3-6.1. While the categories and metadata fields of the MACE application profile are agreed upon, the MACE consortium is still working on the formalization of the values for several metadata fields. A focus is, for the moment, on the classification values of the MACE application profile. The update of the MACE application profile is reported in the half-yearly progress reports of the MACE project.

Prototype description

The prototype is a distributed, services oriented architecture with software both on the content provider side as on the metadata "manager" side.

On the content provider side, software is installed to perform two operations. It provides a mapping from the provider metadata format to the IEEE LOM format following the MACE application profile. Secondly, it allows the content providers to provide an OAI-PMH target, which complies with the OAI-PMH 2.0 specifications[1].

The central "metadata manager" sets up a central harvester. This service has several purposes. The main purpose is of course to harvest all the content metadata from the different content providers and storing it in a central database, called the metadata store. The metadata store's primary database is an XML database, so no transformations have to be performed to store or retrieve the XML metadata.

Additionally, the harvester adds global unique identifiers (see appendix A) for both the metadata and the learning objects described by the metadata set. This ensures that the metadata identifier is unique throughout the whole MACE system and thus identifies the metadata easily. The harvester also adds the time stamp of the point in time when harvesting

[1] http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

took place along with the repository form where the metadata set has been harvested. With this information it is easy to trace back the origin of the metadata.
When the metadata is stored into one metadata store for backup reasons, it is also copied to the second metadata store that provides the actual MACE metadata store. In the MACE metadata store, the metadata is merged with new or enriched metadata, without losing the original harvested metadata (as it is stored in the backup store.)
To increase performance of serving incoming queries, the MACE metadata store is enhanced with a Lucene[1] index that complements the XML database. Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching. Lucene itself is just an indexing and search library and does not contain crawling and HTML parsing functionality. When metadata is inserted, the metadata is both stored in the XML database as well as parsed and transformed into a Lucene document. The Lucene index then enables direct PLQL[2] level 0 and PLQL level 1 queries so that the bottleneck of XQuery translation (needed to enable PLQL level 0 and level 1 on XML databases) is avoided.
The Figure 14 illustrates the technical setup. The metadata is harvested into the Harvested Metadata Store and duplicated in the MACE metadata store. All metadata present in the MACE Metadata Store can be queried through SQI. When metadata is enriched, it can be pushed into the Enriched Metadata Store using SPI. This metadata is then again duplicated to the MACE Metadata Store. This infrastructure preserves all harvested and enriched metadata separately, but also allows for a unified search through the metadata.
Figure 14 describes specific implementations of the different metadata stores and how the insertions and queries work. Compared to the previous figure, the Harvested MetadataStore consists of the “mace-harvest” web service and the database table “HARVESTED”. Analogue to this, the Enriched Metadata Store contains the “mace-enrich” web service and the database table “ENRICHED”. Finally the MACE Metadata Store covers the “mace-ws” web service, the database table “METADATASTORE” and the Lucene index. The communication between the web services and the xml database happens through a JDBC connection. The metadata inserts are translated to SQL insert statements and the high level PLQL queries to XQuery statements. Communication to the Lucene index happens through the Lucene API for both the lower level PLQL queries as the insertions of the metadata.

[1] http://lucene.apache.org/

[2] http://ariadne.cs.kuleuven.be/lomi/index.php/QueryLanguages_v1.0

The MACE content and domain metadata infrastructure is situated within the ARIADNE infrastructure. The ARIADNE architecture is a modular service-oriented architecture built on the principle that interoperability and extensibility is best achieved by the integration of different interfaces as clearly defined modules. These interfaces interoperate based on a formal definition that is independent of the underlying platform. This definition hides the implementation of a language-specific service.

The interfaces that are specified in this document can be implemented in several bindings, among which are

- WSDL binding for making web service implementations of the API.
- Binding for standalone applications, in several programming languages or platforms among which are Java and C#. We provide WSDL bindings1 that implement the different interfaces.

The ARIADNE architecture is depicted in Figure 12 and supports three types of services: repository, core and support services. Repository services contain

- insert and obtain services for both metadata and content,
- an identifier service that provides capabilities to request new identifiers, and
- a value space service that enables the description of vocabularies and its terms.

Besides repository services, core services are defined that enable:

- to enrich content with automatically generated metadata (SAmgI),
- to disaggregate content into its components (DisAgg),
- to make content available through a standardized interface (FedSearch), and
- to collect metadata from various repositories (Harvesting).

Finally, support services are specified that provide, among others, session management and management of Contextualized Attention Metadata (CAM) which is used in the MACE usage metadata architecture.

The repository architecture consists of the bottom layer of Figure 12, i.e. the MetadataStore, the ContentStore, the IdentifierGenerator and the ValueSpaceStore. These interfaces use the support services that are described above.

The modules in Figure 12 that are surrounded with a black rectangle enable users to query the repository architecture by using the Query Service, for instance, but they can also use the recommendation component to find relevant material.

---

[1] http://ariadne.cs.kuleuven.be/lomi/index.php/SpecificationIndex

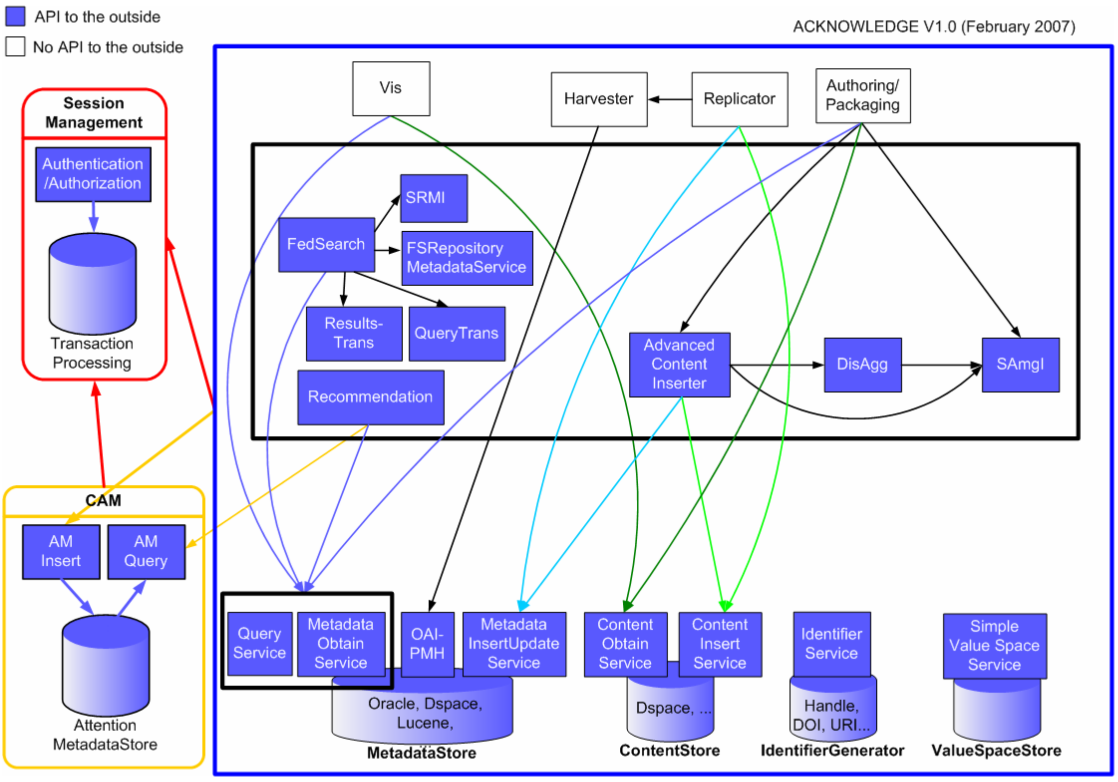


Figure 12 - The ARIADNE architecture

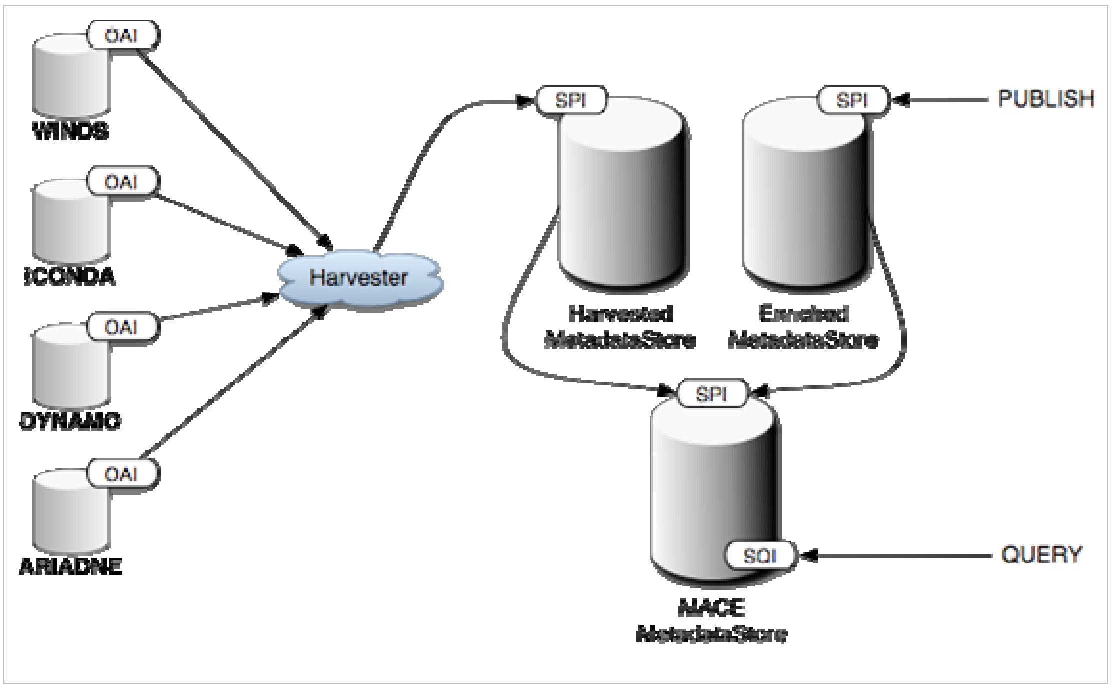


Figure 13 - MACE Content metadata infrastructure

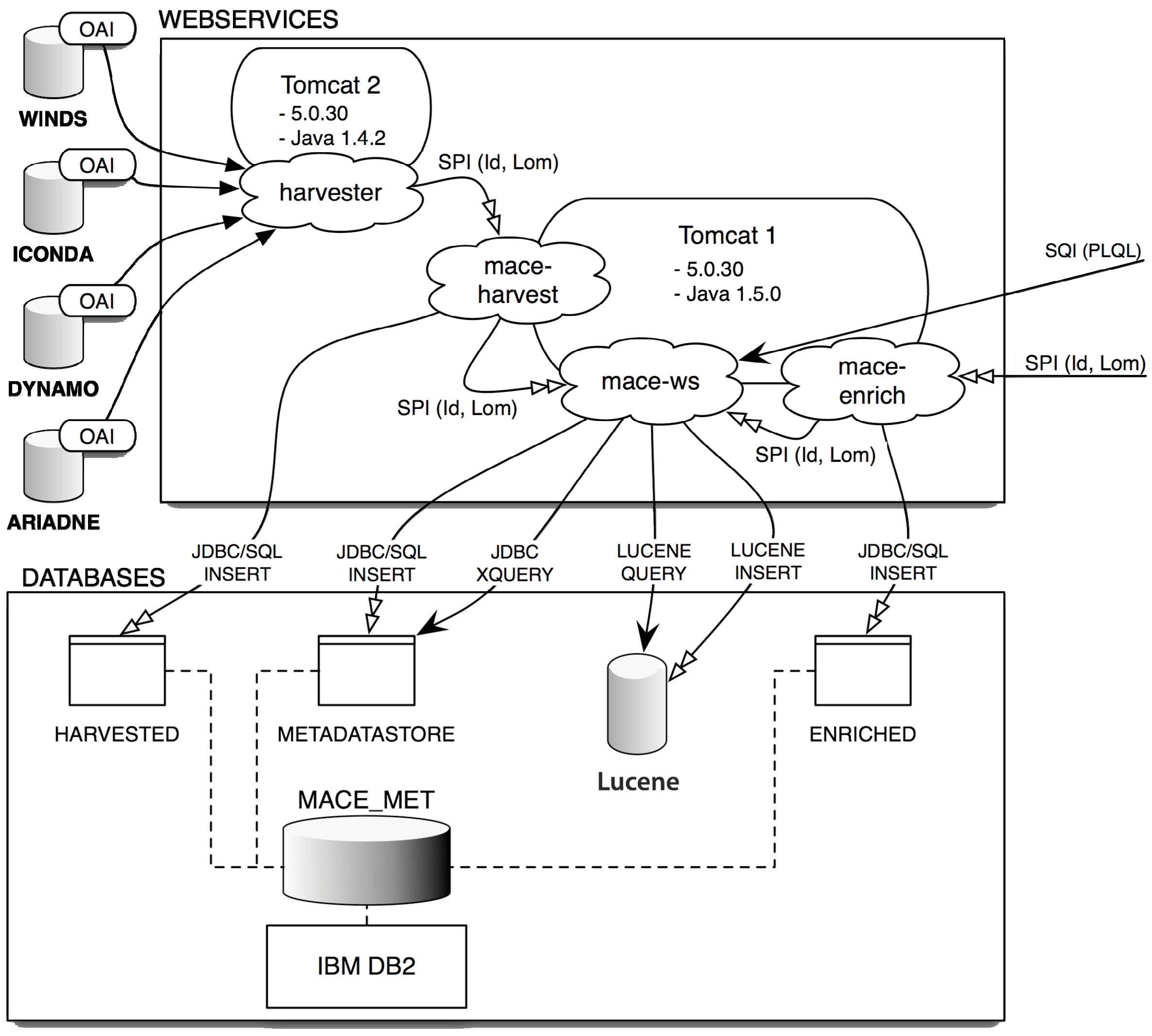


Figure 14 - Technical implementation

Integration into MACE infrastructure

The integration of the above described infrastructure into the MACE architecture bases completely on the usage of web services. These web services are subject to change. Therefore, instead of presenting them here, the reader is referred to the appendix in this document.

Furthermore, the MACE deliverable 7.1-7.3 provides more details on the MACE infrastructure.

Further plans

Apart from the issues to be solved outlined in the MACE deliverable 7.1-7.3, we will continue to integrate further repositories by providing the respective interfaces, support providers in the implementation of the MACE application profile and respective mapping and harvesting software as well as improve performance.

Furthermore, work will also concentrate on the integration of new systems, e.g. the social community system Aloe1 to enable advanced tagging and community building approaches in MACE.

In the context of performance improvements, work will concentrate on better support for PLQL queries, improved storage through reduced injection times of MACE application profile metadata sets, improved interfaces between the stores and other MACE metadata stores, etc.

Last but not least, work will contribute to the issues of MACE user management, user profiling and privacy and security issues respectively.

1 http://aloe-project.de

Conclusion and outlook

This deliverable has shown the prototypes for each enrichment work package. An integration of each prototype into the whole system – the MACE infrastructure as described in joint deliverable JD5 – can be seen now, too. Most of the components will show their full potential when working together to answer requests from end users, which in turn requires the whole consortium to work together on further developing the components and streamlining the interfaces between then. It now also pays that we set on using open standards (AXIS, Web services, XML, SOAP, SQI, SPI) as this will make further work easy.

The next steps are now as follows: We will have the prototypes evaluated, the results being described in joint deliverable JD6 - "Evaluation of functional prototype for metadata tools and concepts".

The evaluation will be followed by a next iteration of requirements (both from the evaluation as well as user requests that did not make it in the first prototype version). After that, the prototypes will be refined and developed further, with a production version planned for M30, to be described in deliverable JD9: "Production version for metadata tools and concepts".

Appendix A: Technology

Widgets

We are planning to implement Widgets for integration into websites to enable MACE Search from different sites.

MACE global identifiers

In order to create unique identifiers across different repositories, we use the OAI identifier schema.
This section explains how to create such identifiers.

NOTE: Both the learning objects as the metadata have MACE GUID's. Be careful what local identifier you use!

OAI identifier specification

See here: http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm

creating MACE GUID's

To create a MACE GUID you need a local identifier (from within a repository) and a repository identifier.
The basic scheme looks like this : oai:<repositoryIdentifier>:<local_Identifier>

example

Here you can find an example from the WINDS repository.

- local identifier : 1008.1
- repositoryIdentifier : winds.gmd.de

--> MACE GUID = oai:winds.gmd.de:1008.1