# D2.1.3 INGESTION SERVICES - 2ND RELEASE

**Advanced Search Services and Enhanced Technological Solutions for the European Digital Library**

Grant Agreement Number: 250527

Funding schema: **Best Practice Network**

Deliverable D2.1.3 WP2.1

Prototype
V.1.3 – March 31 2012
Document. ref.: ASSETS.D2.1.3.CNR.WP2.1.V1.3

| | |
|---|---|
| **Programme Name:** | ICT PSP |
| **Project Number:** | 250527 |
| **Project Title**: | ASSETS |
| **Partners**: | Coordinator: ENG (IT)<br>Contractors: |
| **Document Number**: | D2.1.3 |
| **Work-Package**: | WP2.1 |
| **Deliverable Type:** | Prototype |
| **Contractual Date of Delivery:** | 31-January-2012 |
| **Actual Date of Delivery**: | 31-March-2012 |
| **Title of Document**: | Ingestion Services – 2nd release |
| **Author(s):** | Andrea Esuli (CNR) |
| | Giacomo Berardi (CNR) |
| | Diego Marcheggiani (CNR) |
| | Fabrizio Sebastiani (CNR) |
| | Sergiu Gordea (AIT) |
| | Oscar Täckström (SICS) |
| **Approval of this report** | APPROVED – Luigi Briguglio (ENG) |
| **Summary of this report:** | see Executive Summary |
| **History**: | see Change History |
| **Keyword List**: | ASSETS, Ingestion, Classification, Extraction, Metadata |
| **Availability** | This report is:<br>X public |

## Change History

| Version | Date | Status | Author (Partner) | Description |
|---|---|---|---|---|
| 0.1 | 03/01/2012 | Draft | AE (CNR) | Initial draft, by integrating contributions from ASSETS wiki and dev. environment |
| 0.5 | 08/03/2012 | Draft | AE (CNR) | Release for internal review |
| 0.6 | 12/03/2012 | Review Ready | SG (AIT) | Added the last information about packaging and configuration |
| 1.0 | 23/03/2012 | Pre Final | AE (CNR) | Feedback from reviewers |
| 1.1 | 29/03/2012 | Pre Final | SG (AIT) | Check and revision from Technical Director |
| 1.2 | 30/03/2012 | Final | AE(CNR) | Final version |
| 1.3 | 31/03/2012 | Final | LB (ENG) | APPROVAL AND RELEASE |

# Table of Contents

# Table of Figures

## Executive Summary

This document[1] contains the revised and final specification, technical documentation, and user documentation for the services developed within tasks "T2.1.2 Knowledge extraction", "T2.1.3 Metadata classification", both of which are under the responsibility of CNR, and task "T2.1.4 Ingestion workflow management and Integration", which is under the responsibility of AIT.

The enrichment services are based on a supervised learning approach, i.e., a learning algorithm is trained on examples of manually annotated records; the learning process generates an enrichment model, which is then used to perform the automatic enrichment. After providing a brief introduction on the Assets enrichment services (Section 1), the scientific background on this process is presented (Section 2).

The enrichment modules are implemented as Web-Services being exposed for remote invocation through their rest Interface. The ingestion workflow service connects to them through their client interfaces and provides the users with a web interface to perform the training and automatic enrichment of metadata collections. The software requirements and the technical implementation details are reported in Sections 3 and 4, while Section 5 contains the user manual. Section 6 reports the experimental results aimed at determining the quality of the automatic enrichment process and the guidelines used for the creation training sets, as well.

1 Part of the content of this deliverable already appears in Deliverable 2.0.4 "The ASSET APIs" and in Deliverable 2.1.1 "Specification of Ingestion Services' delivered at M12.

# 1. Introduction

The objective of WP2.1 is to implement a set of services that provide automatic enrichment of metadata records to the ASSETS platform.

The developed services allow ASSETS professional users to:

(i) automatically identify and annotate, within metadata records, pieces of text that denote relevant entities (T2.1.2 “knowledge extraction from metadata records”)

(ii) automatically classify the metadata records according to a set of categories, possibly organized into a taxonomy, relevant for the domain (T2.1.3 “metadata classification”).

The invocation of the services is integrated into the ingestion management tools developed in collaboration with Europeana and “The European Library” (TEL). The mentioned tools will support the back office processes in both these institutions (T2.1.4 “ingestion workflow management”).

These tasks are made complex by the presence of different content providers, within the ASSETS consortium and within Europeana, which have been concerned with different types of content (i.e. text, image, audio, video) and different languages (i.e. there are 27 languages used in Europeana by now). There is a need thus to implement the above-mentioned services in a way that addresses this diversity of content providers, content types, and languages, and in a way that possibly allows new content providers, with new content types described by metadata expressed in new languages, to be also addressed with a minimum additional effort.

As a consequence, the services have been developed according to a supervised learning methodology. Essentially, this means that a new content provider will be able to set up a system for enriching its own metadata by providing to the system a “training” set of enriched metadata records. The system would use these enriched metadata records as indications, or examples, of what enriching metadata records from this content provider means, and would then generate an “automatic enricher” of metadata records provided by this content provider. This mechanism allows to set up automatic metadata enrichers for any type of content provider, any type of content, and any language; of course, adequate training sets of manually enriched metadata records must be given as input.

This supervised learning metaphor underlies all three services tackled within WP2.1. However, its algorithmic realization for the considered services is different, since the individual tasks are different in nature. For instance, T2.1.3 is a task that supports the enrichment of metadata records as a whole by classifying them and will be tackled via automatic text classification technologies. On the other hand, T2.1.2 is a task that supports the enrichment of metadata records not by annotating the full record, but by annotating individual sequences of words within the record. Therefore the knowledge extraction makes use of automatic sequence learning (“information extraction”) technologies.

The integration of the enrichment service execution in a unified workflow (UIM) is achieved web based technologies within the scope of T2.1.4. In this document, we will focus on the description of both the GUI interface. The UIM, as the technical infrastructure used for workflow execution and plug-in orchestration have been described in the deliverable D2.1.1 SPECIFICATION OF INGESTION SERVICES.

The rest of the document contains a brief scientific background to the machine learning method on which the services are based, the description of software requirements for the services, their technical specification, the user manual which describes the format for training data and how to use the test interface, and the evaluation of the quality of the services, based on objective cross-validation experiments.

# 2. Scientific background

This section gives background information about the machine learning methods adopted for the implementation of the enrichment services.

## 2.1 Knowledge Extraction from Metadata Records

T.2.1.2 has to do with automatically annotating the text of which metadata records consist of, by tagging specific parts of this text according to a pre-specified set of words that denote concepts of interest in the domain the metadata records and the corresponding context they refer to. This task is usually referred to as *information extraction* (IE) or *knowledge extraction* in the literature [Ben-Dov and Feldman, 2010, McCallum 2005, Sarawagi 2008]. In other words, the information extraction is the discipline concerned with the extraction of natural language expressions from free text, where these expressions instantiate concepts of interest in a given domain. If there are *n* different concepts of interest, information extraction is a bit like highlighting the text via *n* highlight markers of *n* different colours. For instance, given a corpus of job announcements, one might want to extract from each announcement the natural language expressions that describe the nature of the job, the promised annual salary, the job location, etc.

Another very popular instance of IE is searching free text for *named entities*, i.e., names (or mentions) of persons, locations, geopolitical organizations, and so on [Nadeau and Sekine, 2007]. Put yet another way, IE may be seen as the activity of populating a structured information repository (such as a relational database, where "job", "annual salary", "job location" are attributes) from an unstructured information source such as a corpus of free text. As such, IE is important for enriching digital libraries by making implicit semantics explicit, and is a prerequisite for *concept normalization* ( i.e., linking the mention of a concept to an entry of a controlled vocabulary so that different linguistic manifestations of the same concept link to the same controlled vocabulary entry).

There are two main approaches to designing an IE system. The former is the *rule- based approach*, which consists of manually writing a set of rules which relate natural language patterns with the concepts to be extracted from the text. This approach, while potentially effective, is too costly, since it requires a lot of human effort for writing the rules, which must be jointly written by a domain expert and a natural language engineer. In T2.1.2 we followed the alternative approach, which is based on *supervised machine learning*.

According to this approach, a general-purpose learning software learns to relate natural language patterns with the concepts to be instantiated, from a set of manually annotated free texts, i.e., texts in which the instances of the concepts of interest have been marked by a domain expert. The most important advantage of this approach is that the human effort required for annotating the texts needed for training the system is less than the one needed for manually writing the extraction rules. After all, this is just a manifestation of the fact, well-known in the cognitive sciences, that defining a concept *intensionally* (i.e., specifying a set of rules for recognizing the instances of this concept – say, a set of rules for recognizing red objects) is cognitively much harder for a human that defining the same concept *ostensively* (i.e., pointing to a set of instances of the concept – say, pointing to a set of red objects). A consequence of the machine learning approach is that a system for information extraction may be easily updated to reflect new needs. For example the addition of a new concept into the group of concepts to be identified, or the replacement of one concept set with a completely different one. While the rule-based approach would require in these cases the manual update of the extraction rules via the joint work of a knowledge engineer and a domain expert, the machine learning approach just requires the provision of new training examples annotated according to the new concepts of interest.

In T2.1.2 this is extremely advantageous since the ASSETS consortium (and also the group of Europeana content providers) comprises a variety of content providers coming from libraries, museums, audio-visual archives, etc. They are owning different types of content (and thus likely requiring the annotation of text according to different concepts of interest) and describe it via metadata records formulated in different languages. In the rule-based approach this diversity would entail the need to tackle each combination of <content provider + type of content + language> individually, by manually writing rules for each such combination, while in the machine learning approach each such combination may be tackled by simply providing appropriate training examples.

In the following sections we will first give a formal definition of information extraction and a brief description of “conditional random fields”, the supervised learning algorithm that we have adopted for T2.1.2. Conditional random fields have widely been studied, and are widely used in information extraction applications, ranging from named entity recognition [Zeng et al., 2009], to the analysis of medical reports [Esuli et al., 2011], to medical record anonymisation [Szarvas et al, 2007], and even word hyphenation [Trogkanis and Elkan, 2010]. We will then give a detailed description of the evaluation protocol that we have followed in order to ascertain how accurately the system performs on the metadata records of the ASSETS and Europeana content providers.

## 2.1.1 A formal definition of information extraction

Let a text $U = \{t_1 < s_1 < \ldots < s_{n-1} < t_n\}$ consist of a sequence of *tokens* (i.e., word occurrences) $t_1, \ldots, t_n$ and *separators* (i.e., sequences of blanks and punctuation symbols) $s_1, \ldots, s_{n-1}$, where ”<” means “precedes in the text”. We use the term *textual unit* (or simply *t-unit*), with variables $u_1, u_2, \ldots$, to denote either a token or a separator. Let $C=\{c_1, \ldots, c_m\}$ be a predefined set of *tags* (aka *labels,* or *classes*), or *tagset*. Let $A=\{\sigma_{11}, \ldots, \sigma_{1k}, \ldots, \sigma_{m1}, \ldots, \sigma_{mk}\}$ be an *annotation* for $U$, where a segment $\sigma_{ij}$ *for U* is a pair $(st_{ij},et_{ij})$ composed of a start token $st_{ij} \in$ U and an end token $et_{ij} \in U$ such that $st_{ij} \leq et_{ij}$ (“≤” obviously means “either precedes in the text or coincides with”). Here, the intended semantics is that, given segment $□_{ij}=(st_{ij}, et_{ij}) \in$ A, all t-units between $st_{ij}$ and $et_{ij}$, extremes included, are tagged with tag $c_i$.

Given a universe of texts $\mathcal{U}$ and a universe of segments $\mathcal{A}$, we define *information extraction* (IE) as the task of estimating an unknown target function $□ : \mathcal{U} X C \rightarrow \mathcal{A}$, that defines how a text $U \in \mathcal{U}$ ought to be annotated (according to a tagset C) by an annotation A $\in \mathcal{A}$. The result $□(□): \mathcal{U} X C \rightarrow \mathcal{A}$ of this estimation is called a tagger. Consistently with most mathematical literature we use the caret symbol □() to indicate estimation. Note that the notion of IE we have defined allows a given t-unit to be tagged by more than one tag, and is thus dubbed multi-tag IE. The multi-tag nature of our definition essentially means that, given tagset $C=\{c_1, \ldots, c_m\}$, we can split our original problem into m independent subproblems of estimating a target function $□_i : \mathcal{U} \rightarrow \mathcal{A}_i$ by means of a tagger $□(□_i) : \mathcal{U} \rightarrow \mathcal{A}_i$, for any $i \in \{1, \ldots, m\}$. Likewise, the annotations we will be concerned with from now on will actually be *c*-annotations, i.e., sets of *c<sub>i</sub>-segments* of the form $A_i = \{\sigma_{i1}, ..., \sigma_{ij}\}$. Hereafter we will often drop the prefix $c_i$- when the context makes it implicit.

### 2.1.2 Conditional random fields

As a learning algorithm we have used conditional random fields[Lafferty et al, 2001, Sutton and McCallum, 2007]. Conditional random fields are graphical models that model a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$, in which the variable $\boldsymbol{y}=\langle y_1,..., y_t\rangle$ represents the labels to be predicted, and the variable $\boldsymbol{x}=\langle x_1,..., x_t\rangle$ represents the observed knowledge. In our case ***y*** are the tags to be assigned to the tokens and separators in the text, and ***x*** is the information about these tokens and separators that we will input to the system.

Conditional random fields are often used in classification tasks in which the entities to be classified have highly dependent features (sequence labelling, IE, etc.). Conditional random fields differ from other graphical models, such as Hidden Markov Models, that use a joint probability distribution $p(\boldsymbol{y},\boldsymbol{x})$ and therefore require to know the prior probability distribution $p(\boldsymbol{x})$. In conditional random fields the input variables ***x*** do not need to be represented, thus avoiding the non-trivial modelling of the prior probability distribution $p(\boldsymbol{x})$ and allowing the use of rich and dependent features of the input.

CRF++ is the implementation of *linear-chain conditional random fields*, that define the conditional probability of ***y*** given ***x*** as:

$$P\langle y \mid x : \theta \rangle = \frac{1}{Z(x)} \exp\left( \sum_{t=1}^{T} \sum_{k=1}^{K} \theta_k f_k (y_{t-1}, y_t ; x_t) \right)$$

Where: $Z(x)$ is a normalization factor and $\theta_k$ is one of the $K$ model parameter weights corresponding to a feature function $f_k(y_{t-1}, y_t ; \boldsymbol{x}_t)$.

Each feature function $f_k$ describes the sequence ***x*** at position $t$ with label $y_t$ observed with a transition from label $y_{t-1}$ to $y_t$.

CRF++ allows defining feature functions $f_k$ by using information about the token to be labelled and about the tokens around the token to be labelled. This is possible by defining the size of the window of tokens to be considered around the one to be labelled. The window can be composed by information belonging to tokens that precede the token to be labelled or belonging to tokens that follow the token to be labelled. Having a wide window is important in tasks that require identifying long annotated sequence of tokens. For more details about conditional random fields see [Sutton and McCallum, 2007].

A conditional random field learner needs each t-unit either in a training document or in a test document to be represented in vectorial form. In this work we have used a set of features consisting of the original token as it appears in the text, its part of speech, and the relative lemma, plus information about capitalization, prefixes, suffixes and stemming. To give the learner more robustness over typographical and orthographical errors, we use as features:

- the token lemma,
- the token prefixes (the first character of the token, the first two, the first three, the first four)
- the token suffixes (the last character of the token, the last two, the last three, the

last four),

- the token stem,
- and token capitalization information.

With token capitalization we identify 4 types of capitalization: “all capital”, indicating that all the letters in the word are uppercased, “first letter capital”, indicating that just the first letter of the word is uppercased and the rest of the letters are all lowercased, “all lower”, indicating that none of the letters in the word are uppercased, and “mixed case”, indicating that there are some uppercased letters and some lowercased letters. We also include as a feature the part of speech of the token.

As the evaluation measure we use the recently proposed *token & separator $F_1$ model* [Esuli and Sebastiani, 2010]. According to this model, a tagger is evaluated according to the well-known $F_1$ measure on an event space consisting of all t-units in the text. In other words, each t-unit $u_k$ (rather than each segment, as in the traditional “segmentation F-score” model) counts as a true positive, true negative, false positive, or false negative for a given tag $c_i$, depending on whether $u_k$ belongs to $c_i$ or not in the predicted annotation and in the true annotation. As argued by Esuli and Sebastiani [2010], this model has the advantage that it credits a system for partial success and it penalizes both overtagging and undertagging.

As the well-known $F_1$ metric combines the contributions of precision ($\pi$) and recall ($\rho$) and is defined as $F_1 = \frac{2\pi\rho}{\pi+\rho} = \frac{2TP}{2TP+FP+FN}$ , where *TP*, *FP*, and *FN* stand for the numbers of true positives, false positives, and false negatives, respectively. Note that $F_1$ is undefined when TP=FP=FN =0; in this case we take $F_1$ to equal 1, since the tagger has correctly tagged all t-units as negative.

We compute $F_1$ across the entire test set, i.e., we generate a single contingency table by putting together all t-units in the test set, irrespective of the text they belong to. We then compute both micro-averaged $F_1$ (denoted by $F_1^{\mu}$) and macro-averaged $F_1$ ($F_1^{M}$). $F_1^{\mu}$ is obtained by (i) computing the tag-specific values $TP_i$, $FP_i$ and $FN_i$, (ii) obtaining TP as the sum of the $TP_i$’s (same for *FP* and *FN*), and then (iii) applying the $F_1 = \frac{2TP}{2TP+FP+FN}$ formula. $F_1^{M}$ is obtained by first computing the tag-specific $F_1$ values and then averaging them across the $c_j$’s.

An advantage of using $F_1$ as the evaluation measure is that it is symmetric, i.e., its values do not change if one switches the roles of the human annotator and the automatic tagger. This means that $F_1$ can also be used as a measure of agreement between any two annotators/taggers, regardless of whether they are human or machine, since it does not require one to specify who among the two is the “gold standard” against which the other needs to be checked. For this reason, in the following section we will use $F_1$ both (a) to measure the agreement between our system and the human annotators, and (b) to measure the agreement between the two human annotators. This will allow us to judge in a direct way how far our system is from human performance.

### 2.1.3 References

Ben-Dov, M., Feldman, R.: Text Mining and Information Extraction. In Oded Maimon, Lior Rokach (Eds.): Data Mining and Knowledge Discovery Handbook, 2nd ed. Springer, 2010, pp. 809-835

Esuli, A., Marcheggiani, D., Sebastiani, F.,: Information Extraction from Radiology Reports. Presented at the 7th Italian Conference on Digital Libraries, Pisa, Italy, 2011

Esuli, A., Sebastiani, F.: Evaluating information extraction. In: Proceedings of the Conference on Multilingual and Multimodal Information Access Evaluation (CLEF'10), Padova, IT (2010) 100–111

Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th International Conference on Machine Learning (ICML'01), Williamstown, US (2001) 282–289

Li, L., Zhou, R., Huang, D.: Two-phase biomedical named entity recognition using CRFs. Computational Biology and Chemistry 33(4):334-338 (2009)

McCallum, A.: Information extraction: Distilling structured data from unstructured text. Queue 3(9) (2005) 48–57

Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Linguisticae Investigationes 30(1) (2007) 3—26

Sarawagi, S.: Information extraction. Foundations and Trends in Databases 1(3) (2008) 261--377

Sutton, C., McCallum, A.: An introduction to conditional random fields for relational learning. In Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. The MIT Press, Cambridge, US (2007) 93–127

Szarvas, G., Farkas, R., and Busa-Fekete, R.: State-of-the-art anonymisation of medical data with an iterative machine learning model/framework. Journal of the American Medical Informatics Association, 14(5):574–580, 2007.

Trogkanis, N., Elkan, C.: Conditional Random Fields for Word Hyphenation. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden, 2010, pp. 366-374.

Zeng, G., Zhang, C., Xiao, Bo., Lin, Z.: CRFs-Based Chinese Named Entity Recognition with Improved Tag Set. Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering, 2009, Los Angeles, US, 2009:519-522

## 2.2 Automatic Classification of Metadata Records

As part of their routine information management protocols, many organizations and content providers classify their content (or the metadata that describe these contents) according to a set of categories (or "classification scheme") that effectively describe the domain this content is about. There is often the case that, unless the domain is trivial in nature, this classification scheme has a hierarchical structure, since a non-hierarchical, flat structure would be too clumsy to accommodate the high number of categories that describe the domain. We will indeed assume that content providers do structure their content according to a hierarchically shaped classification scheme. This assumption is non-restrictive, since a flat classification scheme may also be seen as a hierarchical classification scheme consisting of only two levels, the root (level 0) and all the categories (level 1) appended to the root as children.

The field of supervised learning that tackles the classification of textual items (as metadata records are) under hierarchically structured classification schemes is called *hierarchical text categorization* (HTC). Notwithstanding the fact that most large-sized classification schemes for text (e.g. the ACM Classification Scheme, the MESH thesaurus, the NASA thesaurus) indeed have a hierarchical structure, the attention of text classification (TC) researchers has mostly focused on algorithms for “flat” classification. These algorithms, once applied to a hierarchical classification problem, are not capable of taking advantage of the information inherent in the class hierarchy, and may thus be suboptimal, in terms of efficiency and/or effectiveness. On the contrary, many researchers have argued that by leveraging on the hierarchical structure of the classification scheme, heuristics of various kinds can be brought to bear that make the classifier more efficient and/or more effective. This is the reason why, for the purposes of T2.1.3, we have focused our attention on algorithms explicitly devised for HTC.

An important intuition that underlies HTC algorithms is that by viewing classification as the identification of the paths that start from the root, funnel the document down to the subtrees where it belongs (in “Pachinko machine” style), entire other subtrees can be pruned from consideration. That is, when the classifier corresponding to an internal node outputs a negative response, the classifiers corresponding to its descendant nodes do not need to be invoked any more, thus reducing the computational cost of classifier invocation exponentially [Chakrabarti et al. 1998; Koller and Sahami 1997].

A second important intuition is that, by training a binary classifier for an internal node category on a well-selected subset of training examples of local interest only, the resulting classifier may be made more attuned to recognizing the subtle distinctions between documents belonging to that node and those belonging to its sibling nodes. While this technique promises to bring about more effective classifiers, it is also going to improve efficiency, since a smaller set of examples is used in training, thereby making classifier learning speedier. Many of these intuitions have been used in close association with several learning algorithms; the most popular choices in this respect have been naïve Bayesian methods, neural networks, support vector machines and example-based classifiers.

In T2.1.3 we have used an HTC algorithm based on *boosting* technology, called TreeBoost.MH [Esuli et al, 2008]. The reasons for this choice include the fact that TreeBoost.MH has proved to be highly efficient highly accurate and above all competitive algorithms we tested in several applications. We have previously applied this technology for the classification of newswire reports [Esuli et al, 2008], medical discharge reports [Esuli et al, 2008] and radiology reports [Baccianella et al, 2011]. TreeBoost.MH is a multi-label (ML) HTC algorithm that consists of a hierarchical variant of AdaBoost.MH [Schapire and Singer, 2000], the most important member of the boosting algorithms family. Here, *multi-label* (ML) means that a document can belong to zero, one, or several categories at the same time. TreeBoost.MH embodies several intuitions that had arisen before HTC ( e.g. the intuitions that both feature selection and the selection of negative training examples should be performed “locally” (i.e. by paying attention to the topology of the classification scheme). TreeBoost.MH also incorporates the intuition that the weight distribution that boosting algorithms update at every boosting round should likewise be updated “locally”. All these intuitions are embodied within TreeBoost.MH in an elegant and simple way, i.e. by defining TreeBoost.MH as a recursive algorithm that uses AdaBoost.MH as its base step, and that recurs over the tree structure.

In the next two sections we give a concise description of TreeBoost.MH.

### *2.2.1 TreeBoost.MH: A hierarchical version of AdaBoost.MH for multi-label TC*

When discussing an HTC application it is always important to specify what the *semantics of* *the hierarchy* is (i.e., to specify the semantic constraints that a supposedly perfect classifier would enforce). Knowing which constraints are in place has important consequences on which algorithms we might want to apply to this task, and more importantly, on how we should evaluate these algorithms. For instance, one should specify whether a document can in principle belong to zero, one, or several categories (which is indeed our assumption within T2.1.3), or whether it always belongs to one and only one category. No less importantly, one should specify whether it is the case that:

1. a document *d* that is a positive example of a category is also a positive example of all its ancestor categories. We assume this to be the case.
2. a document *d* can in principle be a positive example of an internal node category and at the same time *not* be a positive example of any of its descendant categories. We assume this to be the case.

Assumption 2 is indeed useful for tackling datasets in which documents with these characteristics do occur, while at the same time not preventing us to deal with datasets with the opposite characteristics. A consequence of these two assumptions is that the set of the positive training examples of a non-leaf category is a (possibly proper) superset of the union of the sets of positive training examples of all its descendant categories.

TreeBoost.MH embodies several intuitions that had arisen before within HTC.

The first, fairly obvious intuition (which lies at the basis of practically all HTC algorithms proposed in the literature) is that, in a hierarchical context, the classification of a document is to be seen as a descent through the hierarchy, from the root to the (internal or leaf) categories where the document is deemed to belong. In ML classification this means that each non-root category has an associated binary classifier which acts as a “filter” that prevents unsuitable documents to percolate to the descendants of the category. All test documents that a classifier deems to belong to a category are passed as input to all the binary classifiers corresponding to its children categories, while the documents that the classifier deems not to belong to the category are “blocked” and analysed no further. Note that it may well be the case that a document is deemed to belong to a category by its corresponding classifier and is then rejected by all the binary classifiers corresponding to its children categories; this is indeed consistent with assumption (2) above. In the end, each document may thus reach zero, one, or several (leaf or internal node) categories, and is thus classified as belonging to them.

The second intuition is that the training of a classifier should be performed “locally”, i.e. by paying attention to the topology of the classification scheme. To see this, note that, during classification, if the classifier for a category has performed reasonably well, the classifier for the children categories will only (or mostly) be presented with documents that belong to the subtree rooted in that category. As a result, the training of a classifier for a given category should be performed by using, as negative training examples, the positive training examples of its sibling categories, with the obvious exception of the documents that are also positive training examples of the category itself. In particular, training documents that only belong to categories other than those mentioned above need not be used. The rationale of this choice is that the negative training examples thus selected are “quasi-positive” examples of the category [Fagni and Sebastiani, 2010], i.e. are the negative examples that are closest to the boundary between the positive and the negative region of the category (a notion akin to that of “support vectors” in SVMs), and are thus the most informative negative examples that can be used in training. This is beneficial also from the standpoint of (both training and classification time) efficiency, since fewer training examples and fewer features are involved.

The third intuition is similar, i.e. that feature selection should also be performed “locally”, by paying attention to the topology of the classification scheme. As above, if the classifier for the category has performed reasonably well, the classifiers for its children categories will only (or mostly) be presented with documents that belong to the subtree rooted in the category itself. As a consequence, for the classifiers corresponding to the children categories, it is cost-effective to employ features that are useful in discriminating (only) among themselves. The features that discriminate among categories lying outside the subtree rooted in the category are too general and the features that discriminate among the subcategories of the children categories are too specific. This intuition, albeit in the slightly different context of single-label classification was first presented in [Koller and Sahami, 1997].

TreeBoost.MH also embodies the novel intuition that the weight distribution that boosting algorithms update at every boosting round should likewise be updated “locally”. In fact, the two previously discussed intuitions indicate that hierarchical ML classification is best understood as consisting of several independent (flat) ML classification problems, one for each internal node of the hierarchy. In a boosting context, this means that several independent distributions, each one “local” to an internal node, should be generated and updated by the process. In this way, the “difficulty” of a category will only matter *relative* to the difficulty of its sibling categories. This intuition is of key importance in allowing TreeBoost.MH to obtain exponential savings in the cost of training over AdaBoost.MH.

TreeBoost.MH incorporates these four intuitions by factoring the hierarchical ML classification problem into several “flat” ML classification problems, one for every internal node in the tree. TreeBoost.MH learns in a recursive fashion, generating a binary classifier for each non-root category, by means of which hierarchical classification can be performed in “Pachinko machine” style.

Learning in TreeBoost.MH proceeds by first identifying whether a leaf category has been reached, in which case nothing is done, since the classifiers are generated only at internal nodes. If an internal node has been reached, a ML feature selection process may (optionally) be run to generate a reduced feature set on which the ML classifier for the node will operate. This may be dubbed a “glocal” feature selection policy, since it takes an intermediate stand between the well-known “global” policy (in which the same set of features is selected for all the categories) and the “local” policy (in which a different set of features is chosen for each different category). The glocal policy selects a different set of features for each (maximal) set of sibling categories. We use information gain as the feature selection function and Forman’s [2004] round robin as a feature score globalization method. After the reduced feature set has been identified, TreeBoost.MH calls upon AdaBoost.MH to solve a ML (flat) classification problem for the set of sibling categories. Again, in order to implement the “quasi-positive” policy discussed above, the negative training examples of a category are taken to be the set of the positive training examples of its sibling categories minus the positive training examples of the category itself. Note that this implements the view of several independent, “local” distributions being generated and updated during the boosting process.

Finally, after the ML classifier for a maximal set of sibling categories has been generated, for each such category a recursive call to TreeBoost.MH is issued that processes the subtree rooted in the category in the same way. The final result is a hierarchical ML classifier in the form of a tree of binary classifiers, one for each non-root node, each consisting of a committee of decision stumps.

### *2.2.2 Related work*

HTC was first tackled in Wiener et al. [1995], in the context of a TC system based on neural networks and latent semantic indexing. The intuition that it could be useful to perform feature selection locally by exploiting the topology of the tree is originally due to Koller and Sahami [1997]. However, this work was dealing with single-label text categorization, which means that feature selection was performed ‘‘collectively’’( i.e., relative to the set of children of each internal node). Given that in T2.3.1 we are in an ML classification context, we instead do it ‘‘individually’’ (i.e. relative to each child of any internal node). The intuition that the negative training examples for training the classifier for a given category could be limited to the positive training examples of categories topologically close to it is due to Ng et al. [1997] and Wiener et al. [1995]. The fact that in an ML classification context the classifiers at internal nodes act as ‘‘routers’’ informs much of the HTC literature and is explicitly discussed in Ruiz and Srinivasan [2002], which proposes a HTC system based on neural networks.

Other works in hierarchical text categorization have focused on other specific aspects of the learning task. For instance, the ‘‘shrinkage’’ method presented in McCallum et al. [1998] is aimed at improving parameter estimation for data-sparse leaf categories in a single-label HTC system based on a naive Bayesian method. The underlying intuitions are specific to naive Bayesian methods and do not easily carry over to other contexts. Incidentally, the naive Bayesian approach seems to have been the most popular among HTC researchers, since several other HTC models are hierarchical variations of naive Bayesian learning algorithms [Chakrabarti et al. 1998; Gaussier et al. 2002; Toutanova et al. 2001; Vinokourov and Girolami 2002]. SVMs have also recently gained popularity in this respect [Cai and Hofmann 2004; Dumais and Chen 2000; Liu et al. 2005; Yang et al. 2003].

### *2.2.3 References*

Baccianella, S., Esuli, A., & Sebastiani, F. (2011). Single-Label Classification of Radiology Reports under the ACR Classification Scheme. Presented at the 7th Italian Research Conference on Digital Libraries, Pisa.

Cai, L., & Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM’04), pp. 78–87.

Chakrabarti, S., Dom, B. E., Agrawal, R., & Raghavan, P. (1998). Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. Journal of Very Large Data Bases, 7(3), 163–178.

Dumais, S. T., & Chen, H. (2000). Hierarchical classification of web content. In Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR’00) (pp. 256–263). Athens, GR.

Esuli, A., Fagni, T., & Sebastiani, F. (2008). Boosting Multi-label Hierarchical Text Categorization. Information Retrieval, 11(4):287-313.

Fagni, T. & Sebastiani, F. (2010). Selecting Negative Examples for Hierarchical Text Classification: An Experimental Comparison. Journal of the American Society for Information Science and

Technologies, 61(11):2256-2265.

Forman, G. (2004). A pitfall and solution in multi-class feature selection for text classification. In Proceedings of the 21st International Conference on Machine Learning (ICML'04). Banff, CA.

Gaussier, E., Goutte, C., Popat, K., & Chen, F. (2002). A hierarchical model for clustering and categorising documents. In Proceedings of the 24th European Colloquium on Information Retrieval Research (ECIR'02) (pp. 229–247). Glasgow, UK.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. In Proceedings of the 14th International Conference on Machine Learning (ICML'97) (pp. 170–178). Nashville, US.

Liu, T. Y., Yang, Y., Wan, H., Zeng, H. J., Chen, Z., & Ma, W. Y. (2005). Support vector machines classification with a very large-scale taxonomy. SIGKDD Explorations, 7(1), 36–43.

McCallum, A. K., Rosenfeld, R., Mitchell, T. M., Ng, A. Y. (1998). Improving text classification by shrinkage in a hierarchy of classes. In Proceedings of the 15th International Conference on Machine Learning (ICML'98) (pp. 359–367). Madison, US.

Ng, H. T., Goh, W. B., Low, K. L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. In Proceedings of the 20th ACM International Conference on Research and Development in Information Retrieval (SIGIR'97) (pp. 67–73). Philadelphia, US.

Ruiz, M., & Srinivasan, P. (2002). Hierarchical text classification using neural networks. Information Retrieval, 5(1), 87–118.

Schapire, R. E., & Singer, Y. (2000). BOOSTEXTER: A boosting-based system for text categorization. Machine Learning, 39(2/3), 135–168.

Toutanova, K., Chen, F., Popat, K., & Hofmann, T. (2001). Text classification in a hierarchical mixture model for small training sets. In Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM'01) (pp. 105–113). Atlanta, US.

Vinokourov, A., & Girolami, M. (2002). A probabilistic framework for the hierarchic organisation and classification of document collections. Journal of Intelligent Information Systems, 18(2/3), 153–172.

Wiener, E. D., Pedersen, J. O., & Weigend, A. S. (1995). A neural network approach to topic spotting. In Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95) (pp. 317–332). Las Vegas, US.

Yang, Y., Zhang, J., & Kisiel, B. (2003). A scalability analysis of classifiers in text categorization. In Proceedings of the 26th ACM International Conference on Research and Development in Information Retrieval (SIGIR'03) (pp. 96–103). Toronto, CA.

# 3. Software Requirements Overview

## 3.1 Knowledge extraction

Europeana metadata contains both structured and unstructured information. Structured information is provided by those metadata fields that identify well-specified type of information (e.g., "date", "creator", "language", etc.). Unstructured information is provided by those metadata fields that act as containers of generic information (e.g., "description").

The aim of the knowledge extraction service (Task 2.1.2) is to provide the ASSETS platform with automatic information extraction functionalities that enable to extract relevant structured information (e.g. names of persons, locations, organizations, from unstructured metadata fields contained in Europeana records).

### *3.1.1 Problem statement*

The presence of relevant information stored only in unstructured fields affects the metadata of almost any Europeana content provider. In these cases, potentially relevant information is not given a proper representation in the Europeana records, but it is mentioned in generic, unstructured, textual fields.

The recognition and extraction into dedicated data structures of relevant information contained in unstructured text fields would improve the Europeana records by supporting the completion and/or correction of metadata fields in the original records. Additionally it supports enriching such records with additional fields and enabling the Europeana users to access and search such additional structured information.

### *3.1.2 Product position statement*

The Knowledge extraction service enables Europeana and Content Providers who need to enrich their content by extracting knowledge from unstructured text to perform automatic extraction, once provided with an example set of manually annotated documents.

Unlike completely manual processing or rule-based annotation systems, the knowledge extraction service allows to process large amounts of data by providing a set of examples without requiring the provider to learn complex rule definitions for rule-based annotations. In our approach, the provider is asked just to annotate the relevant pieces of text for the various types of information to be extracted.

### *3.1.3 Stakeholder Descriptions*

Name: **Content Providers and Europeana**

Description: Any content provider that provides data to Europeana, whose data contains relevant fields for the extraction process ( e.g. textual descriptions) can take benefit of these services. The Europeana ingestion team may use the services for the data already acquired by Europeana.

Responsibilities: These stakeholders are responsible to:

(i) define the annotation schema that identifies the relevant types of information to be extracted;

(ii) select and annotate a set of records, following the annotation schema.

Name: **ISTI-CNR**

Description: The research group at ISTI-CNR that is responsible for the knowledge extraction task.

Responsibilities: This stakeholder is responsible to:

(i) support content providers and Europeana in the process of defining the annotation schema;

(ii) provide the proper linguistic analysis, statistical analysis, and machine learning methods best suited for the extraction task as defined by the annotation schema;

(iii) provide the functionalities to include the generated automatic extractors into the ASSETS ingestion workflow.

### *3.1.4 User Environment*

This service is intended to provide its functionalities to ingestion workflow service of the ASSETS platform without any direct interaction with the user.

### *3.1.5 Feature or Functionality Overview*

The development of an information extraction service for a specific type of information is a process that involves three steps:

1. Definition of an annotation schema for a specific information extraction process. The data provider identifies a relevant type of information to be extracted from its records.
2. Definition of a training set for a specific information extraction process. The data provider produces a training set of manually annotated records following the annotation schema. There is no upper limit to the number of annotated records that could be generated by the content providers. This training set of annotated records is given in input to the automatic knowledge extraction system in order to produce and extraction model.
3. Automatically enrich metadata records by extracting information from unstructured text. The content provider sends to the service a set of non-annotated records and specifies the trained extraction model to be applied. The service returns a set of new instances of the records in which the information that is relevant to the extraction model has been annotated and copied into dedicated metadata fields.

### *3.1.6 System Qualities*

*Usability*: The service will provide an API and the relative documentation for inclusion into the ASSETS platform ingestion workflow.

*Reliability:* The service does not provide at any time critical functionalities. The service provides methods for process progress control for batch processing requests.

*Performance:* Information extraction is a task that is part of the ingestion workflow. Given that this process is executed in back-office as part of the ASSETS platform and also that the knowledge extraction process does not require active user interaction, the performance of the system are not a critical aspect. However, the system is based on state-of-the art algorithms and data structures in order to provide the users with the maximum efficiency. Annotation requests for a single record to be typically complete in 5-10 seconds. Batch processing requests exploits bulk processing of records in order to speed up the annotation process that is expected to be about on order of magnitude faster than online annotation.

*User Interfaces:* The service does not require a user interface.

*Software Interfaces:* The service is accessible within the ASSETS ingestion workflow service by providing a RESTful HTTP interface.

### 3.1.7 System Constraints

The service is developed in Java.

### 3.1.8 System Compliance

*Licensing Requirements:* The service adopts an open source, EUPL-compatible license.

*Legal, Copyright, and Other Notices:* Third party components are a Java virtual machine and several Java libraries (e.g. like Log4J). Any other additional library referred by the service is licensed with an open source, EUPL-compatible license.

### 3.1.9 System Documentation

Javadoc documentation for developers is provided for the service API.

## 3.2 Metadata Classification Service

The aim of the metadata classification service (Task 2.1.3) is to provide the ASSETS platform with the functionality of automated classification of metadata records under a taxonomy of categories of interest.

The classification process consists of linking a record to zero, one, or several categories from a (taxonomically organized) set of predefined categories (aka "classes", or "concepts", or "codes"). The set of predefined categories is called the classification scheme. Classification is thus akin to "populating" a taxonomy with instances of the concepts in the taxonomy.

Europeana records are provided by many different content providers, which may:

(i) not use any classification schema for their data,

(ii) use a very specific classification scheme custom-tailored to specific local purposes of the content provider,

(iii) use a standard well-known classification schema for their data, either general-purpose (e.g., Library of the Congress Subject Headings, LCSH) or discipline-specific (e.g., Medical Subject Headings).

Among these three cases the last one is certainly the preferred one for Europeana.

The metadata classification service enables Europeana and content providers to automatically classify unlabelled metadata records, following a set of general-purpose and/or discipline-specific classification schema.

The ultimate goal of the task is making the searching and browsing experience from the user's view more satisfactory; e.g.:

- user can navigate from record to concept and to other records belonging to same concept or sibling concepts;
- user can restrict search to records belonging to a specific concept;
- user can ask to group the search results according to the concepts they belong to.

### *3.2.1 Problem Statement*

Europeana, Content Providers, and Europeana users, could benefit from having the metadata records properly linked to a set of classes in a general-purpose or discipline-specific taxonomy. Most of Europeana records are currently not structured into general-purpose and/or discipline-specific taxonomies, losing the possibility to search, browse and navigate through records by considering the concepts/classes they belong to. Performing the classification of Europeana records into general-purpose and/or discipline-specific taxonomies would enable new access methods to records based on the concept/classes they belong to.

### *3.2.2 Product Position Statement*

The metadata classification service enables to perform automatic metadata classification of record once provided with an example set of manually classified records, supporting Europeana and Content Providers in the process of enriching their content by classifying their records according to a classification schema.

Instead of adopting a completely manual classification of documents, which requires a large human effort, or a rule-based classification method, which requires the provider to learn complex rule definitions, the metadata classification service allows to process large amounts of data by providing a relatively small set of examples with regard to the size of metadata collections.

### *3.2.3 Stakeholder Descriptions*

Name: **Content Providers and Europeana**

Description: Any content provider that provides data to Europeana and the Europeana ingestion team would be interested to use this service for metadata enrichment.

Responsibilities: These stakeholders will be responsible to:

(i) define the classification scheme to be used for record classification;

(ii) select and manually classify a set of records, following the classification scheme.

Name: **ISTI-CNR**

Description: The research group at ISTI-CNR that is responsible for the metadata classification task.

Responsibilities: This stakeholder will be responsible to:

(i) support content providers and Europeana in the process of defining the classification scheme;

(ii) design and develop the proper linguistic analysis, statistical analysis, and machine learning methods best suited for the classification task, as defined by the classification schema;

(iii) provide the functionalities to include the generated metadata classifiers into the ASSETS ingestion workflow.

### *3.2.4 User Environment*

This service is intended to provide functionalities to other services of ASSETS/Europeana, without any direct interaction with the user.

### *3.2.5 Feature or Functionality Overview*

The development of a classification service for a given classification scheme is a process that involves three steps:

1. Definition of a classification scheme for a specific metadata classification process. The content provider identifies a classification scheme for the classification of its records.
2. Definition of a training set for a specific metadata classification process. The content provider produces a training set of at least one thousand manually classified records following the classification scheme.
3. Classify a record according to a given taxonomy. The data provider sends to the service a set of unclassified records, and specifies the metadata classification model to be adopted. The service returns a set of new instances of the records in which the proper codes are assigned to records.

### *3.2.6 System Qualities*

*Usability:* The service provides an API, and the relative documentation, for inclusion into the ASSETS platform ingestion workflow.

*Reliability:* The service does not provide at any time critical functionalities. The service provides methods for process progress control for batch processing requests.

*Performance:* Performance is not a critical issue for the classification service, since it is performed in the backoffice part of Europeana, and it does not require user interaction. However, the system adopts state-of-the-art algorithms and data structures in order to provide an efficient service. Online classification requests for a single record will typically complete in 2-5 seconds, while bulk requests will be processed about one order of magnitude faster, exploiting bulk processing of records.

*User Interfaces:* The service does not require a user interface.

*Software Interfaces:* The service is accessible within the ASSETS ingestion workflow service by means of a RESTful HTTP interface.

### 3.2.7 System Constraints

The service is developed in Java.

### 3.2.8 System Compliance

*Licensing Requirements:* The service adopts an open source, EUPL-compatible license.

*Legal, Copyright, and Other Notices:* Third party components are a Java virtual machine and several Java libraries (e.g. like Log4J). Any other additional library referred by the service is licensed with on open source, EUPL-compatible license.

### 3.2.9 System Documentation

Javadoc documentation for developers is provided for the service API.

## 3.3 Ingestion Workflow service

### 3.3.1 *Problem statement*

The Europeana web portal implements a search engine over the European cultural heritage. In order to provide this functionality, an index with the description of the masterpieces of objects available in Galleries, Libraries, Archives and Museum (GLAM) institutions was created by aggregating information retrieved from the Content Providers (CPs, see Figure 1).

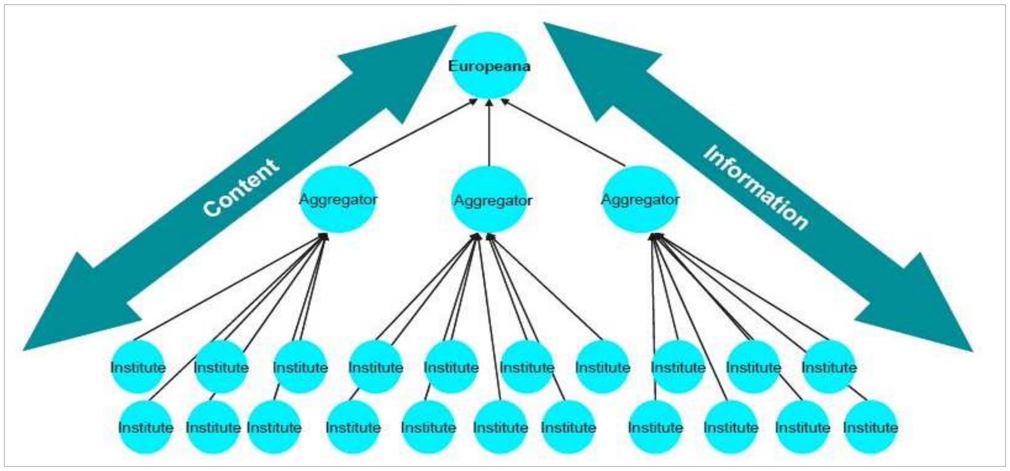


*Figure 1 Aggregators in the Europeana organisation mode*

The aggregation and ingestion are complex processes which were formalized in a flow diagram within the requirements specification for the Danube release of Europeana. A unified ingestion manager application is developed in order to offer support for scheduling, executing and monitoring ingestion related activities. The professional services that have been developed within the ASSETS project address the steps 7.Data Enrichment and 9. AIP-Phase of the process sketched in Figure 2.

Further descriptions of each ingestion related task is available in Europeanalabs:

http://europeanalabs.eu/wiki/SpecificationsDanubeRequirementsContentInTools

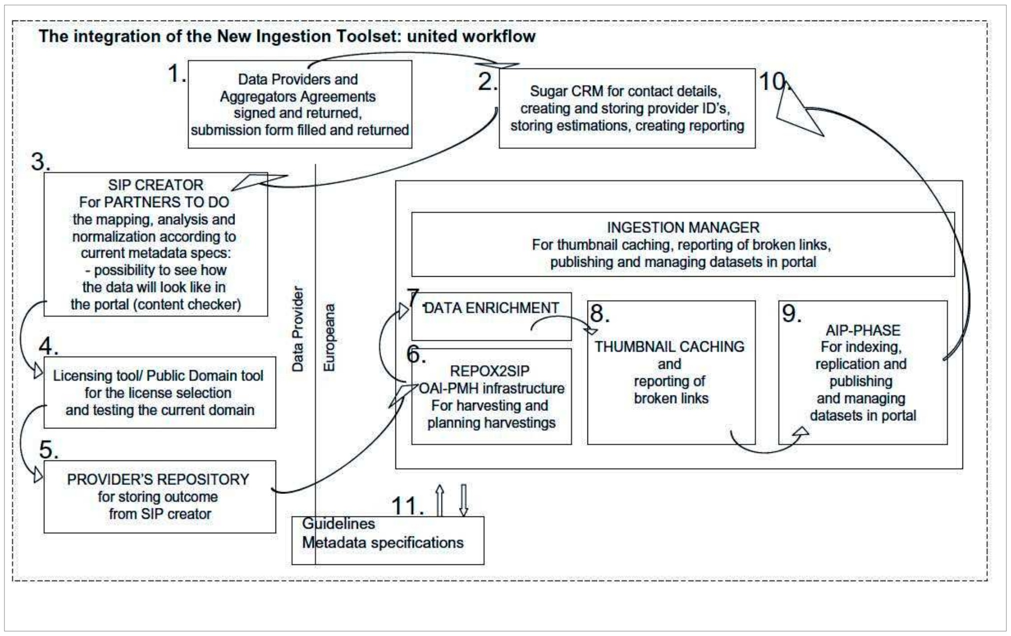

The integration of the New Ingestion Toolset: united workflow
1. Data Providers and Aggregators Agreements signed and returned, submission form filled and returned
2. Sugar CRM for contact details, creating and storing provider ID's, storing estimations, creating reporting
10.
3. SIP CREATOR For PARTNERS TO DO the mapping, analysis and normalization according to current metadata specs: - possibility to see how the data will look like in the portal (content checker)
4. Licensing tool/ Public Domain tool for the license selection and testing the current domain
5. PROVIDER'S REPOSITORY for storing outcome from SIP creator
Data Provider
Europeana
INGESTION MANAGER For thumbnail caching, reporting of broken links, publishing and managing datasets in portal
7 DATA ENRICHMENT
6. REPOX2SIP OAI-PMH infrastructure For harvesting and planning harvestings
8. THUMBNAIL CACHING and reporting of broken links
9. AIP-PHASE For indexing, replication and publishing and managing datasets in portal
11. Guidelines Metadata specifications

*Figure 2 The Europeana Ingestion Process*

### 3.3.2 Functionality overview

The metadata enrichment are the subject of this document, and they belong to the workflow (step 7, Figure 2) while the indexing services and preservation services (Step 9, Figure 2) will be ordinately described in the deliverables D2.2.5 ("Scalable Content Indexing and Ranking") and D2.3.2 ("Deployed Preservation Services").

ASSETS project extends the GUI of the Europeana Ingestion "Control Panel" by implementing screens that manage the following functionalities:

- Enrichment model learning: by using a training set appropriate for their metadata, the content providers or Europeana are allowed to run the learning of enrichment models for metadata classification or knowledge extraction,
- Enrichment by metadata classification: the classification of a collection can be performed by selecting an appropriate classification model,
- Enrichment by knowledge extraction: the extraction of the structured knowledge from the object descriptions can be performed similarly to the classification by selecting an appropriate model and the collection to be enriched.

During the project specification phase, the most of the professional and the indexing services were identified to belong to the post ingestion process. In our case this is equivalent to the Access Information Package creation Phase (AIP-Phase) indicated in step 9 of the Europeana Ingestion Process.

In the DoW, these services were defined to be accessed through REST or command line interfaces. During the integration in Europeana, these services will be bound within the UIM-Control Panel as well.

### 3.3.3 System Qualities

*Usability*: The service will provide a web based graphical UIM and a user manual to support its usage.

*Reliability:* The enrichment of large metadata collections or the model training might take long time, the invocation must not block the GUI of the ingestion workflow management. Moreover, the GUI must display the status of the training/enrichment tasks.

*Performance:* The ingestion process is a back office process, and the response time of the Ingestion Workflow interface must be in the range of regular web applications.

*User Interfaces:* The GUI must be web based and support the most common web browsers (i.e. Internet explorer, Mozilla)

*Software Interfaces:* The Ingestion Workflow must be able to invoke webservices remotely and must be able to work on a cluster infrastructure. The workflow must implement an easy extendable architecture.

# 4. Technical Documentation:

## 4.1 UML Diagrams

### 4.1.1 Knowledge extraction service

***Use case**: Training of a knowledge extraction model*

The Knowledge Extraction Service is based on the use of supervised learning algorithms. In order to allow the service to perform knowledge extraction of a certain type of information, a training set has to be provided as input to the learning algorithm. The training set consists of examples of records in which the relevant information to be extracted has been manually annotated by human experts.

*Actor*: Content provider.

The content provider generates a training set containing manually annotated records in order to support the machine learning process of the knowledge extraction service for relevant information types, e.g., time expressions, named entities (persons, locations, organizations).

*Basic flow of events:*

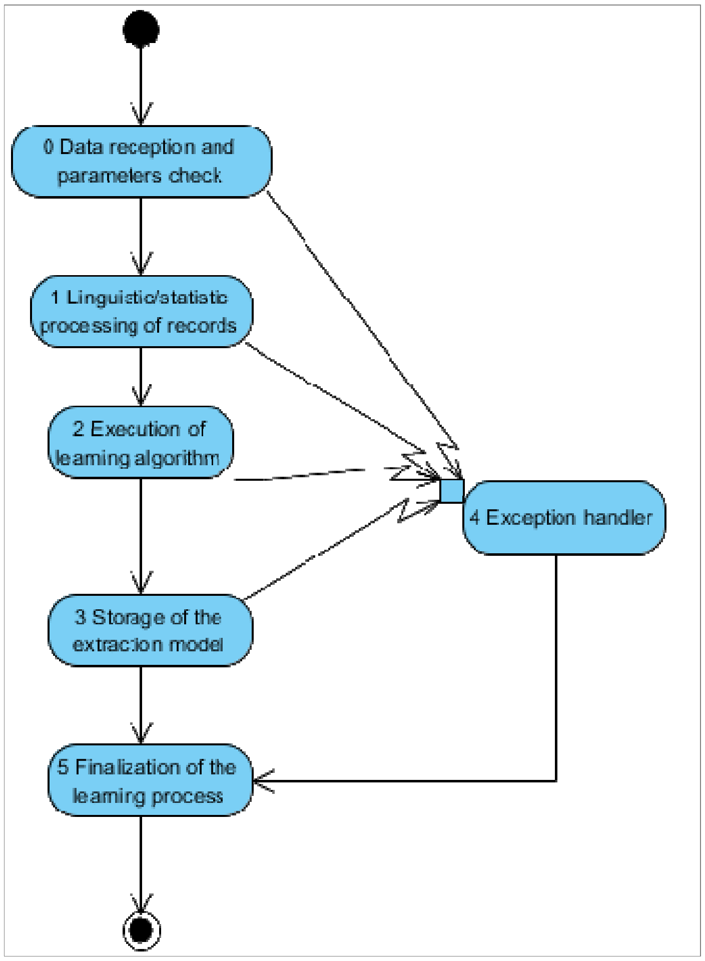

0 Data reception and parameters check
1 Linguistic/statistic processing of records
2 Execution of learning algorithm
3 Storage of the extraction model
4 Exception handler
5 Finalization of the learning process

*Figure 3 Flow of events for the training of the extraction service*

0. The use case begins when the learning algorithm is provided with a training set of examples of annotated records.
1. The relevant textual metadata fields of the records composing the training set are processed by linguistic and statistical tools in order to produce their vectorial representation that will then be processed by the learning algorithm.
2. The learning algorithm processes the vectorial representations and learns a knowledge extraction model.
3. The information relative to the transformation of records into the corresponding vectorial representations and the knowledge extraction model are stored for future use by the knowledge extraction service.
4. In case an error condition happens during the execution, any exception is caught by an exception handler that manages the error in order to guarantee a safe conclusion of the process (e.g., properly releasing the acquired resources).
5. The learning process ends.

*Key scenarios*:

1. Success: correct input parameters and well-formed training set.
2. Error: wrong input parameters or incorrect training set format.

*Post-conditions:*

1. Success: an extraction model is generated and made available to the ingestion workflow.
2. Error: no extraction model is generated and an error message is returned.

***Use case:*** *Knowledge Extraction Service invocation*

The Knowledge Extraction Service is a plugin in the Ingestion workflow service and provides enrichment functionality.

*Actor*: ASSETS ingestion workflow user.

As part of the configuration of the ingestion workflow service, the actor selects the records to be processed by the knowledge extraction service, the proper type of knowledge to be extracted, and after the service execution the actor inspects the results.

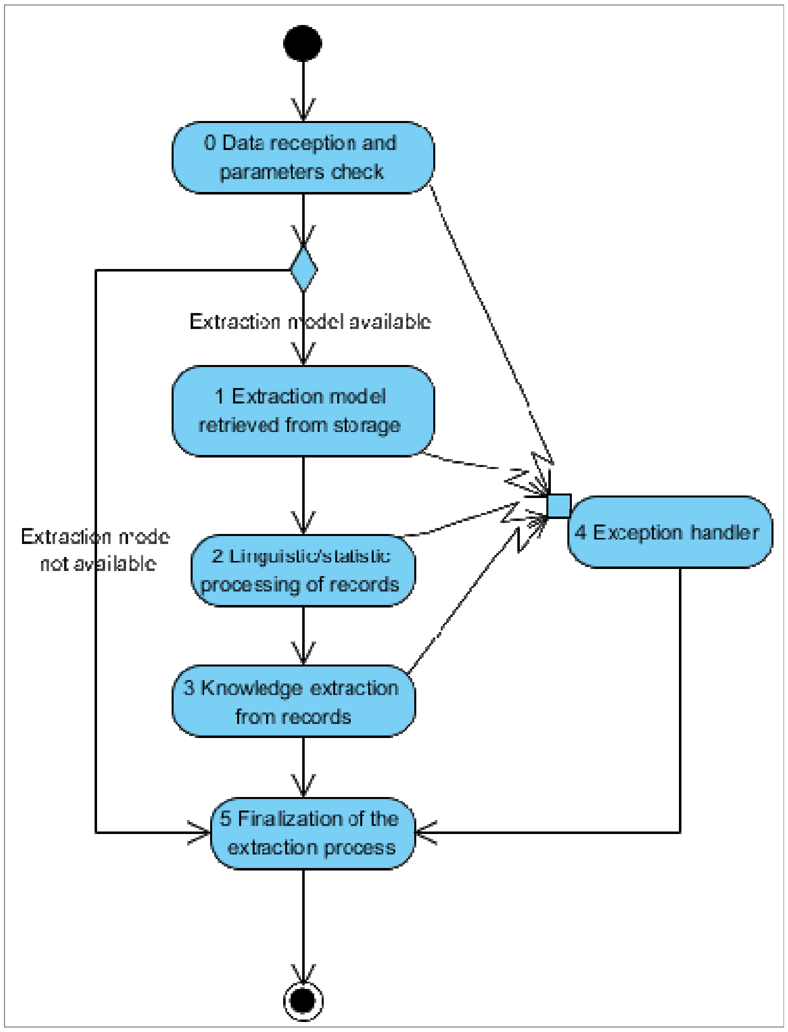

0 Data reception and parameters check
Extraction model available
1 Extraction model retrieved from storage
Extraction model not available
2 Linguistic/statistic processing of records
3 Knowledge extraction from records
4 Exception handler
5 Finalization of the extraction process

*Figure 4 Flow of the events for the enrichment process based on the extraction service*

*Basic flow of events:*

0. The ingestion workflow service notifies the knowledge extraction service about the records that have to be processed, the type of knowledge that has to be extracted from records, and other relevant parameters. Ingestion service sends the records to be processed to the knowledge extraction service. If a knowledge extraction model for the required type of information is not available, the extraction process immediately ends; no output is produced.
1. The proper knowledge extraction model and the information relative to the transformation of records into the corresponding vectorial representations are retrieved from storage.
2. The records provided in input are converted into vectorial representations.
3. Each vectorial representation is processed by the knowledge extraction model, resulting in annotation/extraction of pieces of text from the original record. Such

pieces of text are used to fill in the metadata fields that are designed to store the type of information that is the subject of the extraction process (e.g., author, date).

4. In case an error condition happens during the execution, any exception is caught by an exception handler that manages the error in order to guarantee a safe conclusion of the process (e.g., properly releasing the acquired resources).
5. Once the knowledge extraction service has completed its processing, the ingestion service retrieves the enriched version of the records from the knowledge extraction service.

*Key scenarios*:

1. Success: the input parameters are correct and the requested knowledge extraction model is available.
2. Error: wrong input parameters or the requested knowledge extraction model is unavailable.

*Post-conditions:*

1. Success: an enriched and distinct copy of the input record is made available to the ingestion workflow service.
2. Error: no enriched records are generated.

### *4.1.2 Metadata classification service*

***Use case****: Training of a classification model*

The Metadata Classification Service is implemented by using supervised learning algorithms. In order to allow the service to perform classification of records under a given classification scheme, a training set has to be provided as input to the learning algorithm. The training set consists of examples of records that have been manually classified by human experts.

*Actor*: Content Provider.

The content provider generates a training set in the form of manually classified records, in order to support the training of the automatic metadata classification service for relevant classification schemes.

*Basic flow of events:*

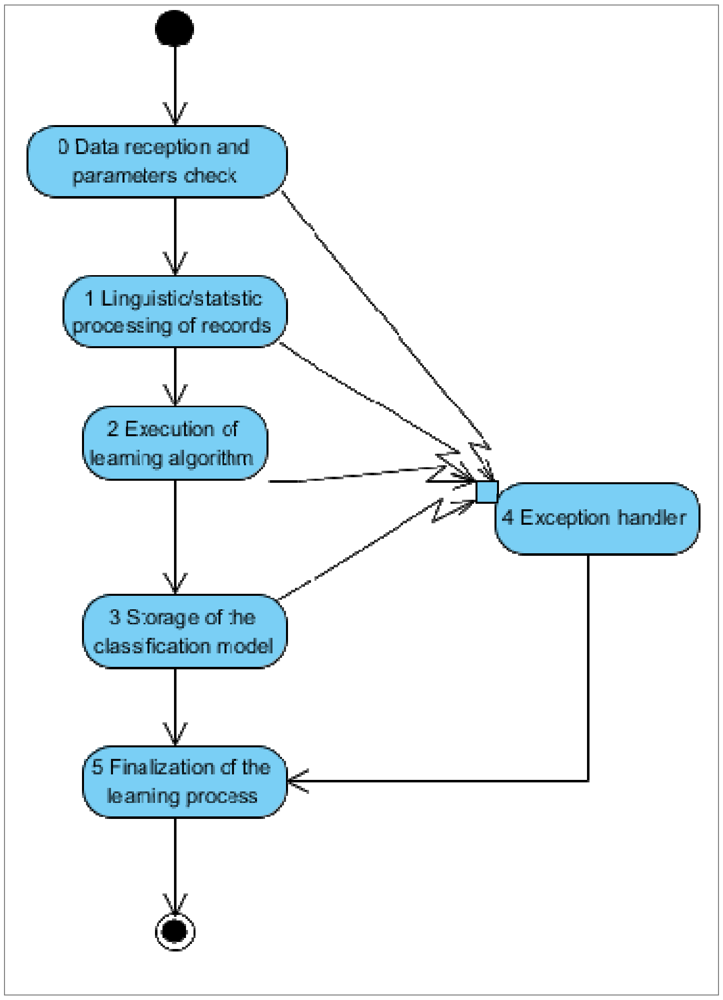

0 Data reception and parameters check
1 Linguistic/statistic processing of records
2 Execution of learning algorithm
3 Storage of the classification model
4 Exception handler
5 Finalization of the learning process

*Figure 5 Flow of events for the training of the classification service*

0. The use case begins when the learning algorithm is provided with a training set of examples of classified records.
1. The textual metadata fields of the records composing the training set are combined and processed by linguistic and statistical tools in order to produce vectorial representation that will then be processed by the learning algorithm.
2. The learning algorithm processes the vectorial representations and learns a classification model.
3. The information relative to the transformation of records into the corresponding vectorial representations and the classification model are stored for future use by the classification service.
4. In case an error condition happens during the execution, any exception is caught by an exception handler that manages the error in order to guarantee a safe conclusion of the process (e.g., properly releasing the acquired resources).
5. The learning process ends.

*Key Scenarios:*

1. Success: correct input parameters and well-formed training set.
2. Error: wrong input parameters or training set in incorrect format.

*Post-conditions:*

1. Success: a classification model is generated and made available to the ingestion workflow service.
2. Error: no classification model is generated.

***Use case:** metadata classification service invocation*

The Metadata Classification Service is a plugin in the Ingestion workflow service and provides the enrichment functionality.

*Actor:* ASSETS ingestion workflow manager (user).

As part of the configuration of the ingestion workflow service, the actor selects the records to be processed by the metadata classification service; the proper classification scheme is then applied, and after the service execution the actor inspects the results.

*Basic flow of events:*

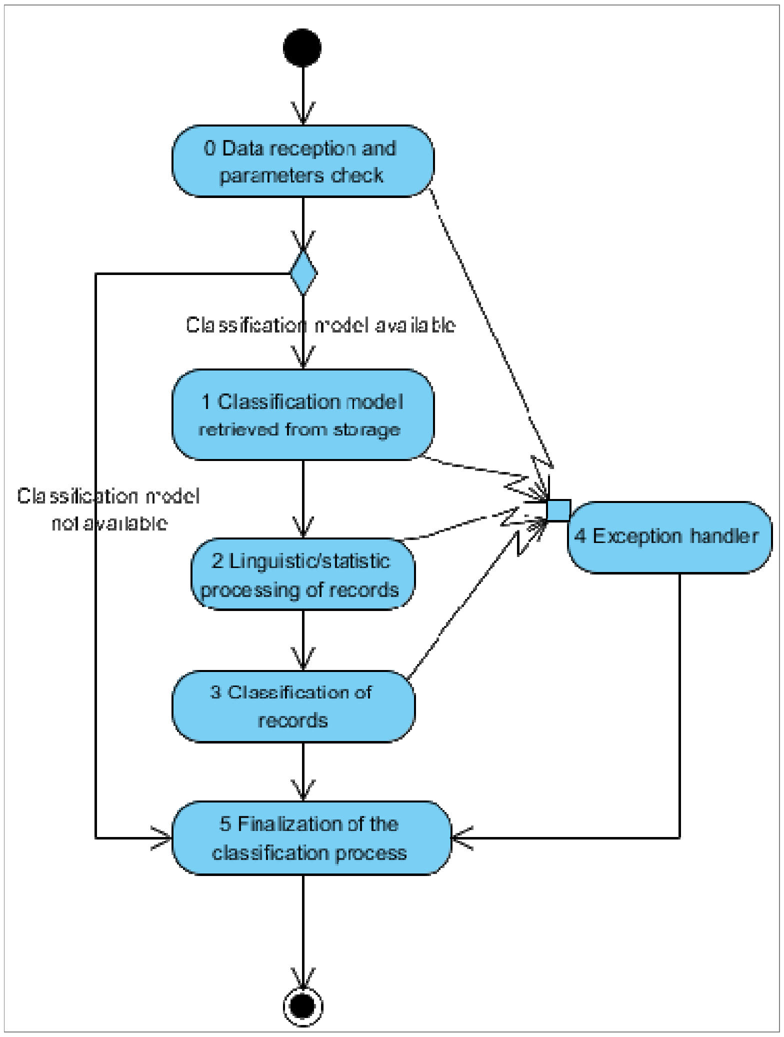

0 Data reception and parameters check
Classification model available
Classification model not available
1 Classification model retrieved from storage
2 Linguistic/statistic processing of records
3 Classification of records
4 Exception handler
5 Finalization of the classification process

*Figure 6 Flow of the events for the enrichment process based on the classification service*

0. The ingestion workflow service notifies the metadata classification service about the records that have to be processed, the classification scheme to be applied, and other relevant parameters. Ingestion workflow service sends the records to be processed to the classification service.
1. The proper classification model and the information relative to the transformation of records into the corresponding vectorial representations are retrieved from storage.
2. The records provided in input are converted into vectorial representations.
3. Each vectorial representation is processed by the classification model, resulting in classification label begin associated to the original record.
4. In case an error condition happens during the execution, any exception is caught by

an exception handler that manages the error in order to guarantee a safe conclusion of the process (e.g., properly releasing the acquired resources).

5. Once the classification service has completed its processing the ingestion service retrieves the classified version of the records from the classification service.

*Key scenarios*:

1. Success: the input parameters are correct and the requested metadata classification model is available.
2. Error: wrong input parameters or the requested metadata classification model is unavailable.

*Post-conditions:*

1. Success:  an enriched and distinct copy of the input record is made available to the ingestion workflow service.
2. Error: no enriched records are generated.

### *4.1.3 Ingestion workflow management*

The execution of the Ingestion Workflow will perform the actions indicated in the activity diagram presented in the Figure 7. This use-case describes the activities which are performed during the invocation of the ingestion workflow. The main goal of this service is to integrate and manage the execution of the metadata enrichments as an integrated process. A (web based) graphical interface was implemented in order to allow the users to perform the following actions:

- start the execution of the ingestion workflow,
- monitor the progress of the execution,
- verify the successful workflow execution,
- visualize error reports.

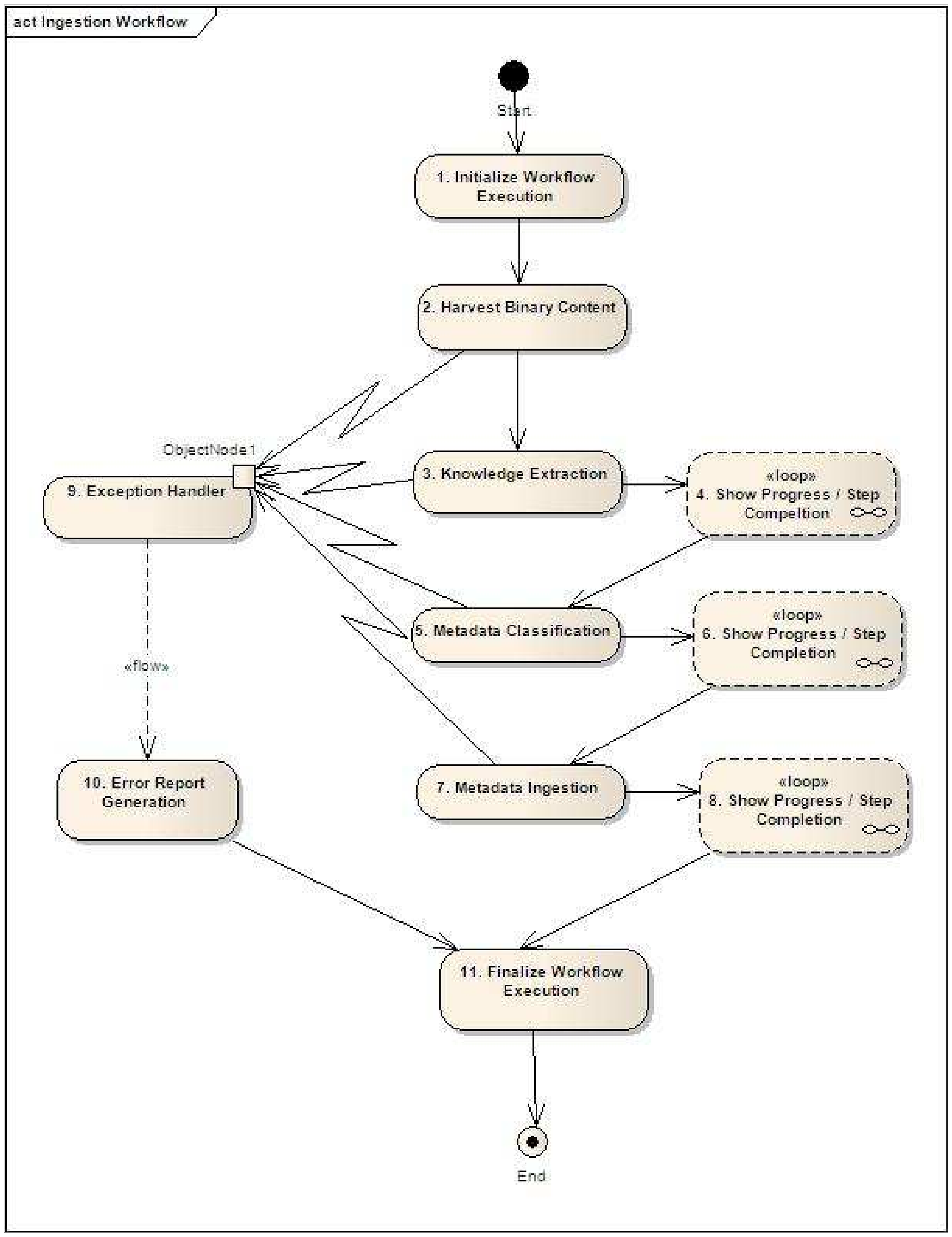

act Ingestion Workflow
Start
1. Initialize Workflow Execution
2. Harvest Binary Content
3. Knowledge Extraction
«loop» 4. Show Progress / Step Compeltion
5. Metadata Classification
«loop» 6. Show Progress / Step Completion
7. Metadata Ingestion
«loop» 8. Show Progress / Step Completion
ObjectNode1
9. Exception Handler
«flow»
10. Error Report Generation
11. Finalize Workflow Execution
End

*Figure 7 Activity diagram for ingestion workflow management*

Basic flow of events:

0. Start. The execution of the use-case begins when the user accessed a corresponding screen in the graphical interface

1. Initialize Workflow Execution. The first step in the workflow will initialize the execution. This step must resolve problems like: loading component configurations, localizing and connecting to the local resources, etc. The ingestion workflow will process a bundle of

objects grouped in a collection.  This step will also allow users to upload their own metadata collections and training sets to the server.

2. Harvest Binary Content. The second step in the execution workflow could optionally invoke the service responsible for harvesting the binary files associated with the given collection (only if a valid reference is available in item's metadata). The files which are not available will be skipped; the broken links will be reported in service logs. If the file was already downloaded in a previous execution, the harvesting will be skipped in order to speed up the process and to avoid overload on the content provision server.

3. Knowledge Extraction. Invocation of the knowledge extraction service. See Knowledge Extraction Service Requirements

4. Show Progress / Step Completion. The most of the services invoked by the ingestion workflow are long lasting processes. Therefore, they will be started asynchronously and the progress of the computations will be provided by request. Therefore the activities Nr. 4-6-8 will be implemented as loop activities, and will permanently indicate the progress of the associated activity.

5. Metadata Classification. Invocation of the metadata classification service. See Metadata Classification Service Requirements

6. Show Progress / Step Completion. See step 4.

7. Metadata Ingestion. The enriched metadata will be stored in the ASSETS/Europeana (backend) database.

8. Show Progress / Step Completion. See step 4.

9. Exception Handler. The invocation of each activity from the ingestion workflow may fail and throw an exception. The exception handler is responsible for extracting the user friendly information from the caught exceptions and passing this information to the next processing step.

10. Error Report generation. In the case that a workflow execution exception occurred, an error report will be generated, stored in the system logs and shown to the user.

11. Finalize Workflow Execution. For either faulty or successful execution, the ingestion workflow must terminate with releasing the locked resources and sending user notification. Eventually, information about the execution of the ingestion workflow can be stored in the database.

## 4.2 Service APIs

### *4.2.1 Knowledge extraction service*

The Knowledge Extraction service exposes its functionalities through three interfaces.

| Service Name | Knowledge Extraction |
|---|---|
| Responsibility | 1. Extraction of structured information from unstructured textual metadata fields within Europeana metadata records |

| | |
|---|---|
| Provided Interfaces | 1. KnowedgeExtractionTrainer,<br>2. KnowledgeExtractionManager,<br>3. KnowledgeExtractor |
| Dependencies | ASSETS common |

The KnowledgeExtractionManager interface enables the management of the available knowledge extraction model.

| | |
|---|---|
| Interface Name | KnowledgeExtractionManager |
| Key Concepts | MetadataKnowledgeExtractionModel, KnowledgeExtractorDescriptor |
| Operations | • listMetadataKnowledgeExtractor: lists the knowledge extractor models available for enrichment.<br>• deleteMetadataKnowledgeExtractor: deletes a knowledge extractor model.<br>• getKnowledgeExtractorDescriptor: returns a descriptor of the knowledge extractor model detailing the type of extracted information. |

The KnowledgeExtractionTrainer interface enables the creation of new extraction models based on a proper formatted training set. It also allows checking the status of a training process.

| | |
|---|---|
| Interface Name | KnowledgeExtractionTrainer |
| Key Concepts | MetadataKnowedgeExtractionTrainingSet, MetadataKnowledgeExtractionModel |
| Operations | • trainMetadataKnowledgeExtractor: learns an extraction model with the provided training data.<br>• getTrainingStatus: returns the status of a learning process. |

The KnowledgeExtraction interface enables the use of a trained extraction model to enrich a metadata record.

| | |
|---|---|
| Interface Name | KnowledgeExtractor |
| Key Concepts | MetadataDataset, MetadataKnoledgeExtractionModel |
| Operations | • extractKnowledgeFromMetadata: enriches a metadata record using a previously trained knowledge extraction model. |

Any training request is assigned with a unique identifier. The status of a training process is described by an enumeration that lists four possible states of a training process (see Figure 8). The type of information extracted by an extraction model is described by a KnowledgeExtractionDescriptor object (see Figure 8).

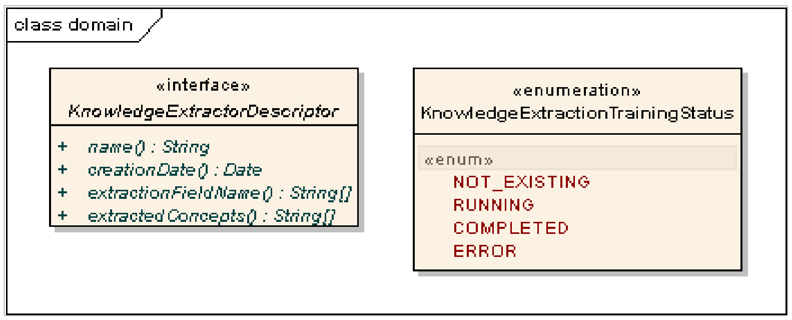

class domain
«interface»
KnowledgeExtractorDescriptor
+ name() : String
+ creationDate() : Date
+ extractionFieldName() : String[]
+ extractedConcepts() : String[]
«enumeration»
KnowledgeExtractionTrainingStatus
«enum»
NOT_EXISTING
RUNNING
COMPLETED
ERROR

*Figure 8  Knowledge Extraction Data Model*

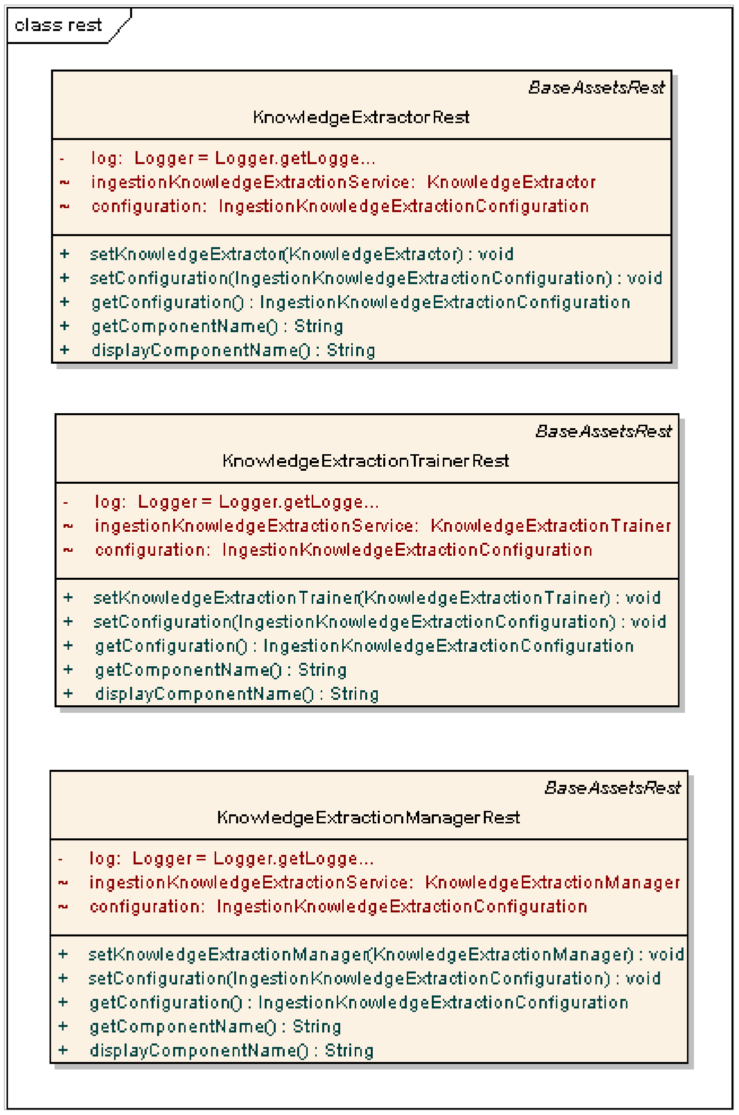

class rest
BaseAssetsRest
KnowledgeExtractorRest
- log: Logger = Logger.getLogge...
~ ingestionKnowledgeExtractionService: KnowledgeExtractor
~ configuration: IngestionKnowledgeExtractionConfiguration
+ setKnowledgeExtractor(KnowledgeExtractor) : void
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ getComponentName() : String
+ displayComponentName() : String
BaseAssetsRest
KnowledgeExtractionTrainerRest
- log: Logger = Logger.getLogge...
~ ingestionKnowledgeExtractionService: KnowledgeExtractionTrainer
~ configuration: IngestionKnowledgeExtractionConfiguration
+ setKnowledgeExtractionTrainer(KnowledgeExtractionTrainer) : void
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ getComponentName() : String
+ displayComponentName() : String
BaseAssetsRest
KnowledgeExtractionManagerRest
- log: Logger = Logger.getLogge...
~ ingestionKnowledgeExtractionService: KnowledgeExtractionManager
~ configuration: IngestionKnowledgeExtractionConfiguration
+ setKnowledgeExtractionManager(KnowledgeExtractionManager) : void
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ getComponentName() : String
+ displayComponentName() : String

*Figure 9 Knowledge Extraction REST API*

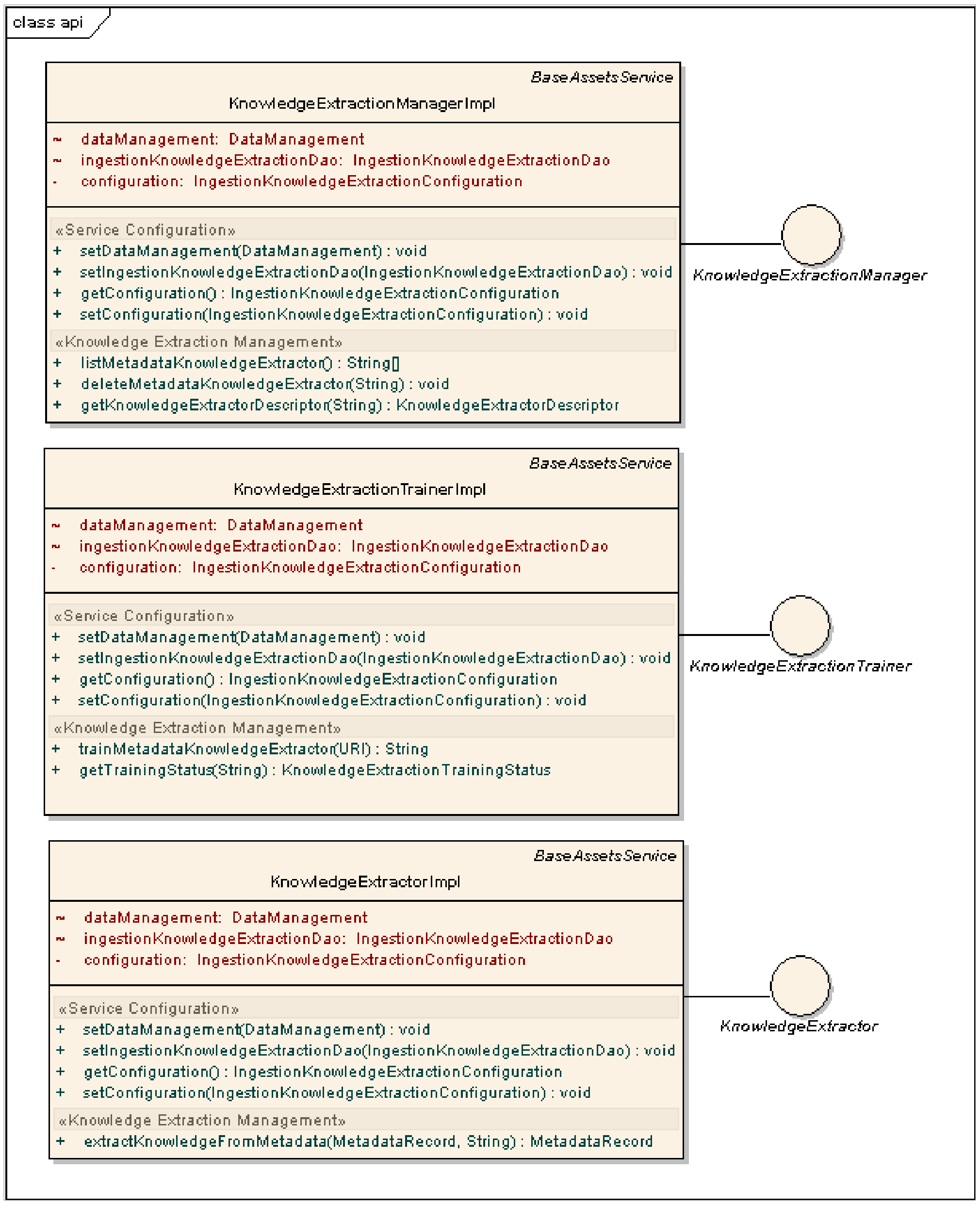

class api
BaseAssetsService
KnowledgeExtractionManagerImpl
~ dataManagement: DataManagement
~ ingestionKnowledgeExtractionDao: IngestionKnowledgeExtractionDao
- configuration: IngestionKnowledgeExtractionConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ setIngestionKnowledgeExtractionDao(IngestionKnowledgeExtractionDao) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
«Knowledge Extraction Management»
+ listMetadataKnowledgeExtractor() : String[]
+ deleteMetadataKnowledgeExtractor(String) : void
+ getKnowledgeExtractorDescriptor(String) : KnowledgeExtractorDescriptor
KnowledgeExtractionManager
BaseAssetsService
KnowledgeExtractionTrainerImpl
~ dataManagement: DataManagement
~ ingestionKnowledgeExtractionDao: IngestionKnowledgeExtractionDao
- configuration: IngestionKnowledgeExtractionConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ setIngestionKnowledgeExtractionDao(IngestionKnowledgeExtractionDao) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
«Knowledge Extraction Management»
+ trainMetadataKnowledgeExtractor(URI) : String
+ getTrainingStatus(String) : KnowledgeExtractionTrainingStatus
KnowledgeExtractionTrainer
BaseAssetsService
KnowledgeExtractorImpl
~ dataManagement: DataManagement
~ ingestionKnowledgeExtractionDao: IngestionKnowledgeExtractionDao
- configuration: IngestionKnowledgeExtractionConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ setIngestionKnowledgeExtractionDao(IngestionKnowledgeExtractionDao) : void
+ getConfiguration() : IngestionKnowledgeExtractionConfiguration
+ setConfiguration(IngestionKnowledgeExtractionConfiguration) : void
«Knowledge Extraction Management»
+ extractKnowledgeFromMetadata(MetadataRecord, String) : MetadataRecord
KnowledgeExtractor

*Figure 10  Knowledge Extraction API*

### 4.2.2 Metadata classification service

The Metadata Classification service exposes its functionalities through three interfaces.

| Service Name | Metadata Classification |
|---|---|
| Responsibility | 1. Classification of Europeana metadata records on relevant taxonomies |
| Provided Interfaces | 1. ClassificationTrainer,<br>2. ClassificationManager,<br>3. ClassificationService |
| Dependencies | ASSETS common |

The ClassificationManager interface enables the management of the available knowledge extraction model.

| Interface Name | ClassificationManager |
|---|---|
| Key Concepts | MetadataClassificationModel |
| Operations | • listMetadataClassifier: lists the available metadata classifier models.<br>• deleteMetadataClassifier: deletes a metadata classifier model. |

The ClassificationTrainer interface enables the creation of new extraction models, based on a proper training set, and the control of the status of a training process.

| Interface Name | ClassificationTrainer |
|---|---|
| Key Concepts | MetadataClassificationTrainingSet, MetadataClassificationModel |
| Operations | • trainMetadataClassifier: learns a metadata classifier model, using the provided training data.<br>• getTrainingStatus: returns the status of a learning process. |

The ClassificationService interface enables the use of a trained extraction model to classify a metadata record under a taxonomy of interest.

| Interface Name | ClassificationService |
|---|---|
| Key Concepts | MetadataDataset, MetadataClassificationModel |
| Operations | • classifyMetadata: classifies a metadata record using a metadata classifier model. |

Any training request is assigned a unique identifier. The status of a training process is described by an enumeration that lists four possible states of a training process. The type of taxonomy applied by a classification model is described by a MetadataClassificationDescriptor object.

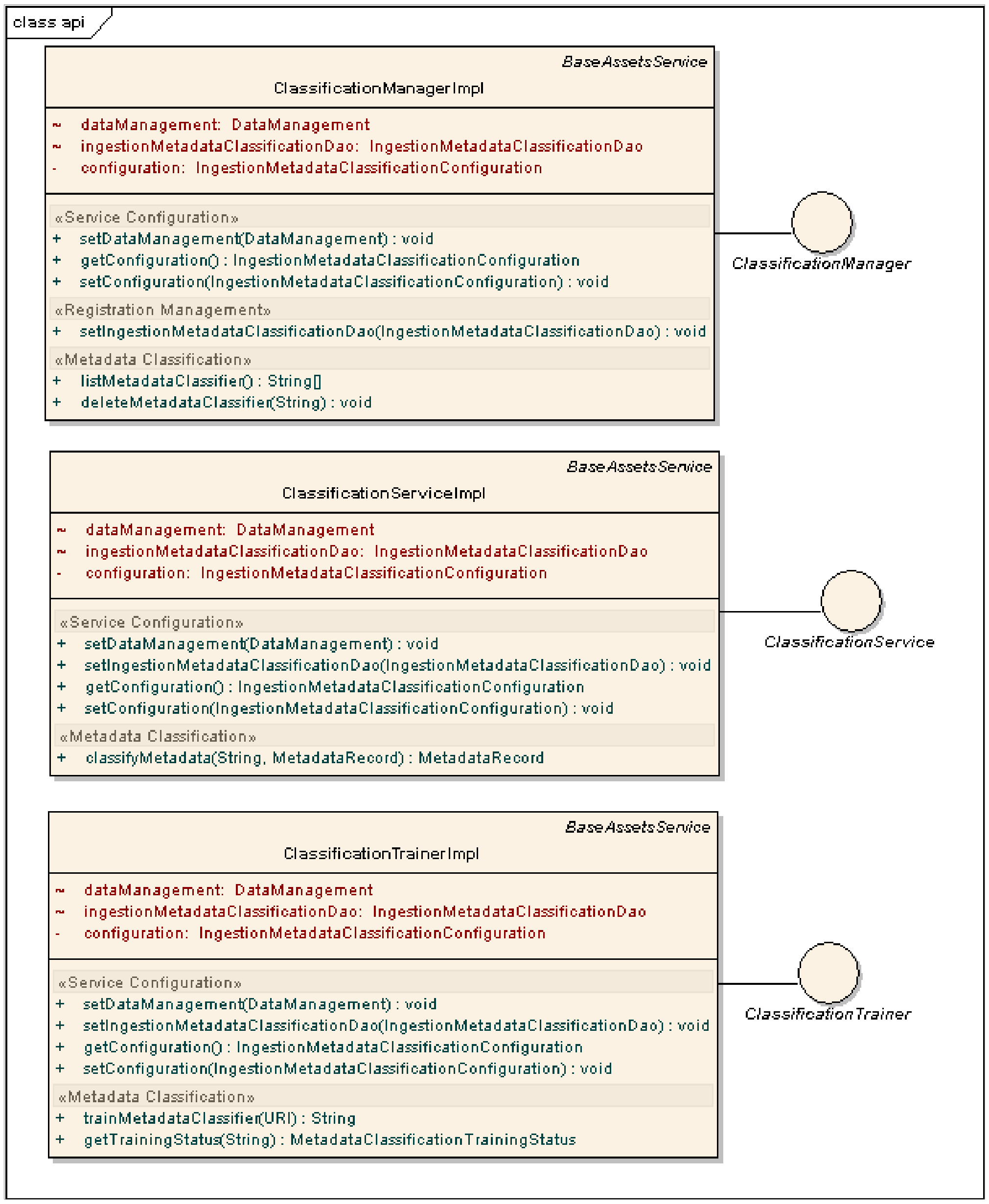

class api
BaseAssetsService
ClassificationManagerImpl
~ dataManagement: DataManagement
~ ingestionMetadataClassificationDao: IngestionMetadataClassificationDao
- configuration: IngestionMetadataClassificationConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
«Registration Management»
+ setIngestionMetadataClassificationDao(IngestionMetadataClassificationDao) : void
«Metadata Classification»
+ listMetadataClassifier() : String[]
+ deleteMetadataClassifier(String) : void
ClassificationManager
BaseAssetsService
ClassificationServiceImpl
~ dataManagement: DataManagement
~ ingestionMetadataClassificationDao: IngestionMetadataClassificationDao
- configuration: IngestionMetadataClassificationConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ setIngestionMetadataClassificationDao(IngestionMetadataClassificationDao) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
«Metadata Classification»
+ classifyMetadata(String, MetadataRecord) : MetadataRecord
ClassificationService
BaseAssetsService
ClassificationTrainerImpl
~ dataManagement: DataManagement
~ ingestionMetadataClassificationDao: IngestionMetadataClassificationDao
- configuration: IngestionMetadataClassificationConfiguration
«Service Configuration»
+ setDataManagement(DataManagement) : void
+ setIngestionMetadataClassificationDao(IngestionMetadataClassificationDao) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
«Metadata Classification»
+ trainMetadataClassifier(URI) : String
+ getTrainingStatus(String) : MetadataClassificationTrainingStatus
ClassificationTrainer

*Figure 11 Metadata Classification Service API*

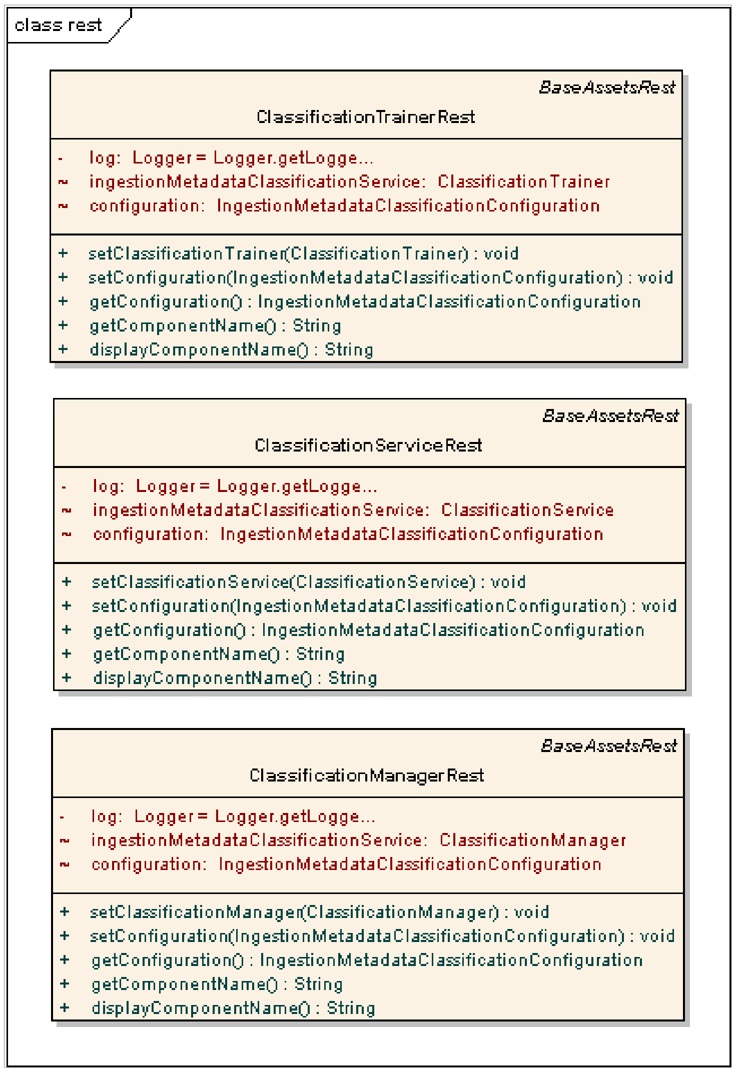

class rest
BaseAssetsRest
ClassificationTrainerRest
- log: Logger = Logger.getLogge...
~ ingestionMetadataClassificationService: ClassificationTrainer
~ configuration: IngestionMetadataClassificationConfiguration
+ setClassificationTrainer(ClassificationTrainer) : void
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ getComponentName() : String
+ displayComponentName() : String
BaseAssetsRest
ClassificationServiceRest
- log: Logger = Logger.getLogge...
~ ingestionMetadataClassificationService: ClassificationService
~ configuration: IngestionMetadataClassificationConfiguration
+ setClassificationService(ClassificationService) : void
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ getComponentName() : String
+ displayComponentName() : String
BaseAssetsRest
ClassificationManagerRest
- log: Logger = Logger.getLogge...
~ ingestionMetadataClassificationService: ClassificationManager
~ configuration: IngestionMetadataClassificationConfiguration
+ setClassificationManager(ClassificationManager) : void
+ setConfiguration(IngestionMetadataClassificationConfiguration) : void
+ getConfiguration() : IngestionMetadataClassificationConfiguration
+ getComponentName() : String
+ displayComponentName() : String

*Figure 12 Metadata Classification REST API*

### 4.2.3 Ingestion workflow management service

The ingestion workflow management service is implemented as a client-server application which provides a rich graphical user interface for invocation of the enrichment services. It is implemented as an extension of the Europeana ingestion control panel and it is implemented using the Google web toolkit (gwt) technology. The details of the GUI implementation are presented in the following tables and UML diagrams.

The following table presents a brief overview of the service and the most important interfaces it uses:

| | |
|---|---|
| Service Name | Ingestion workflow management – Frontend |
| Responsibility | Provides a GUI for performing the enrichment activities |
| Provided Interfaces | AssetsIngestionControlPanel ,<br>EnrichmentServiceProxy, |
| Dependencies | Europeana ingestion framework, knowledge extraction service, metadata classification service |

#### *Assets control panel*

The ASSETS ingestion control panel is the class responsible for the binding of the enrichment screens into the ingestion application. The rendering of the enrichment screens (see Section 5.3) is handled by special classes which extends the GWT Widget class. The overview of the AssetsIngestionPanel class is presented in the following table and the UML diagram presenting the details of these classes is available in Figure 13:

| | |
|---|---|
| Interface Name | AssetsIngestionControlPanel extends EuropeanaIngestionControlPanel |
| Key Concepts | AssetsModelLearningWidget (model learning screen)<br>AssetsEnrichmentTestWidget (enrichment test screen)<br>AssetsCollectionKnowledgeExtractionWidget (knowledge extraction screen)<br>AssetsCollectionClassificationWidget (metadata classification screen) |
| Operations | • onModuleLoad(): enhances the method in the parent class by adding the initialization of the enrichment service<br>• addMenuEntries(): enhances the method in the parent class by binding the enrichment screens in the main menu |

#### *Assets Enrichment Service Proxy*

The enrichment service is in charge of executing the operations requested through the control panel. It provides methods that handle the asynchronous communication between the client (web browser) and the UIM server. The data transferred between client and server is packaged as data transfer objects (DTOs). The enrichment service proxy is responsible for the invocation of the knowledge extraction and metadata classification services and the aggregation of the information displayed in the enrichment screens. The EnrichmentServiceProxy interface is briefly documented in the following table and the full class diagram with dependencies is presented in Figure 14:

| Interface Name | EnrichmentServiceImpl implements EnrichmentServiceProxy |
|---|---|
| Key Concepts | CollectionEnrichmentResultDTO: Object used to group together the information displayed after running the enrichment processes on a collection<br><br>CollectionObjectDTO: Object used to represent the enriched collection object<br><br>CollectionObjectPreviewDTO: Object used to present a preview of a collection object by displaying the most common information.<br><br>EnrichmentModelDTO: Object used to display the properties of the enrichment models in the GUI<br><br>EnrichmentResultDTO: Container object used to keep references to the input and the output of the (testing of the) enrichment process<br><br>EuropeanaCollectionDTO: Object used to represent the basic information related to the metadata collections in the GUI<br><br>ModelLearningStatusDTO: Object used to display the status of the model learning in the GUI<br><br>SearchCriteriaDTO: Object used for collecting the search filter used by users to select the content. |
| Operations | • learnModel(): Method used for invoking the learning the enrichment model from the provided training set<br>• getModelList(): Method used to retrieve the list of the available enrichment models<br>• getColectionList(): Method used for retrieving the list of collections already ingested in Europeana application<br>• getCollectionObjects(): Method used to retrieve the preview of the objects available in a collection with a given id.<br>• getModelStatus(): Checks the learning status for a model with a given id.<br>• deleteModel(): Method used to invoke the deletion of enrichment models<br>• performKnowledgeExtraction(): The method used for invoking the knowledge extraction enrichment for the object identified by a given URI<br>• getDataProviderList():The method used to retrieve the list of the data |

| | |
|---|---|
| | providers for given aggregator name<br>• searchObjects(): Method used to search for objects in Europeana index by using a given search criteria<br>• performKnowledgeExtraction(fileName): Method used to invoke the enrichment for all objects available in a given file<br>• getColectionFileList(): Method used to retrieve the list of the collection files that are already uploaded on the server.<br>• performMetadataClassification(): Method used for performing the metadata classification enrichment of all objects available in a given collection |

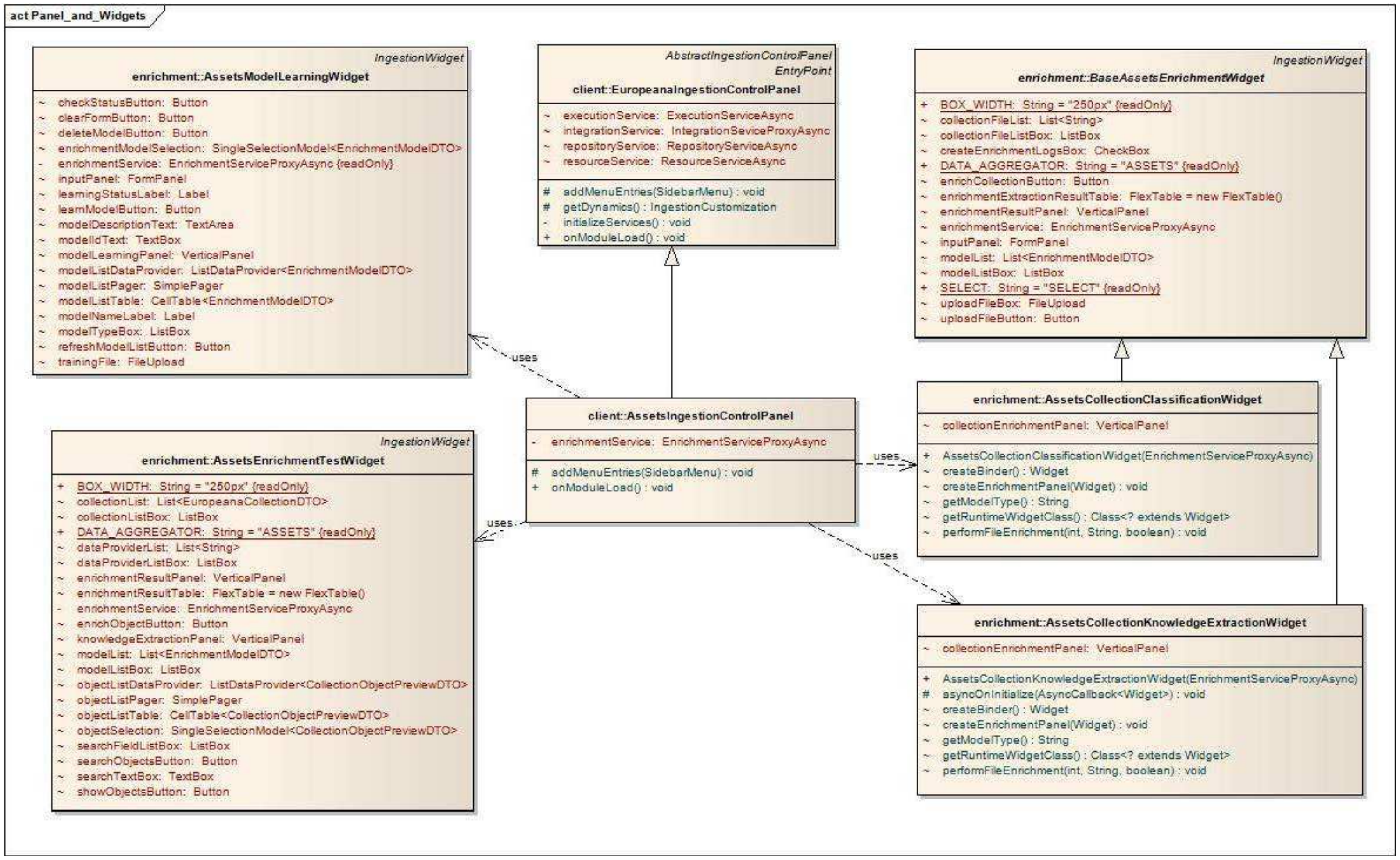

act Panel_and_Widgets
IngestionWidget
enrichment::AssetsModelLearningWidget
~ checkStatusButton: Button
~ clearFormButton: Button
~ deleteModelButton: Button
~ enrichmentModelSelection: SingleSelectionModel<EnrichmentModelDTO>
- enrichmentService: EnrichmentServiceProxyAsync {readOnly}
~ inputPanel: FormPanel
~ learningStatusLabel: Label
~ learnModelButton: Button
~ modelDescriptionText: TextArea
~ modelIdText: TextBox
~ modelLearningPanel: VerticalPanel
~ modelListDataProvider: ListDataProvider<EnrichmentModelDTO>
~ modelListPager: SimplePager
~ modelListTable: CellTable<EnrichmentModelDTO>
~ modelNameLabel: Label
~ modelTypeBox: ListBox
~ refreshModelListButton: Button
~ trainingFile: FileUpload
AbstractIngestionControlPanel
EntryPoint
client::EuropeanaIngestionControlPanel
~ executionService: ExecutionServiceAsync
~ integrationService: IntegrationSeviceProxyAsync
~ repositoryService: RepositoryServiceAsync
~ resourceService: ResourceServiceAsync
# addMenuEntries(SidebarMenu) : void
# getDynamics() : IngestionCustomization
- initializeServices() : void
+ onModuleLoad() : void
IngestionWidget
enrichment::BaseAssetsEnrichmentWidget
+ BOX_WIDTH: String = "250px" {readOnly}
~ collectionFileList: List<String>
~ collectionFileListBox: ListBox
~ createEnrichmentLogsBox: CheckBox
+ DATA_AGGREGATOR: String = "ASSETS" {readOnly}
~ enrichCollectionButton: Button
~ enrichmentExtractionResultTable: FlexTable = new FlexTable()
~ enrichmentResultPanel: VerticalPanel
~ enrichmentService: EnrichmentServiceProxyAsync
~ inputPanel: FormPanel
~ modelList: List<EnrichmentModelDTO>
~ modelListBox: ListBox
+ SELECT: String = "SELECT" {readOnly}
~ uploadFileBox: FileUpload
~ uploadFileButton: Button
uses
client::AssetsIngestionControlPanel
- enrichmentService: EnrichmentServiceProxyAsync
# addMenuEntries(SidebarMenu) : void
+ onModuleLoad() : void
uses
uses
uses
enrichment::AssetsCollectionClassificationWidget
~ collectionEnrichmentPanel: VerticalPanel
+ AssetsCollectionClassificationWidget(EnrichmentServiceProxyAsync)
~ createBinder() : Widget
~ createEnrichmentPanel(Widget) : void
~ getModelType() : String
~ getRuntimeWidgetClass() : Class<? extends Widget>
~ performFileEnrichment(int, String, boolean) : void
IngestionWidget
enrichment::AssetsEnrichmentTestWidget
+ BOX_WIDTH: String = "250px" {readOnly}
~ collectionList: List<EuropeanaCollectionDTO>
~ collectionListBox: ListBox
+ DATA_AGGREGATOR: String = "ASSETS" {readOnly}
~ dataProviderList: List<String>
~ dataProviderListBox: ListBox
~ enrichmentResultPanel: VerticalPanel
~ enrichmentResultTable: FlexTable = new FlexTable()
- enrichmentService: EnrichmentServiceProxyAsync
~ enrichObjectButton: Button
~ knowledgeExtractionPanel: VerticalPanel
~ modelList: List<EnrichmentModelDTO>
~ modelListBox: ListBox
~ objectListDataProvider: ListDataProvider<CollectionObjectPreviewDTO>
~ objectListPager: SimplePager
~ objectListTable: CellTable<CollectionObjectPreviewDTO>
~ objectSelection: SingleSelectionModel<CollectionObjectPreviewDTO>
~ searchFieldListBox: ListBox
~ searchObjectsButton: Button
~ searchTextBox: TextBox
~ showObjectsButton: Button
enrichment::AssetsCollectionKnowledgeExtractionWidget
~ collectionEnrichmentPanel: VerticalPanel
+ AssetsCollectionKnowledgeExtractionWidget(EnrichmentServiceProxyAsync)
# asyncOnInitialize(AsyncCallback<Widget>) : void
~ createBinder() : Widget
~ createEnrichmentPanel(Widget) : void
~ getModelType() : String
~ getRuntimeWidgetClass() : Class<? extends Widget>
~ performFileEnrichment(int, String, boolean) : void

Figure 13  Assets Ingestion Panel API

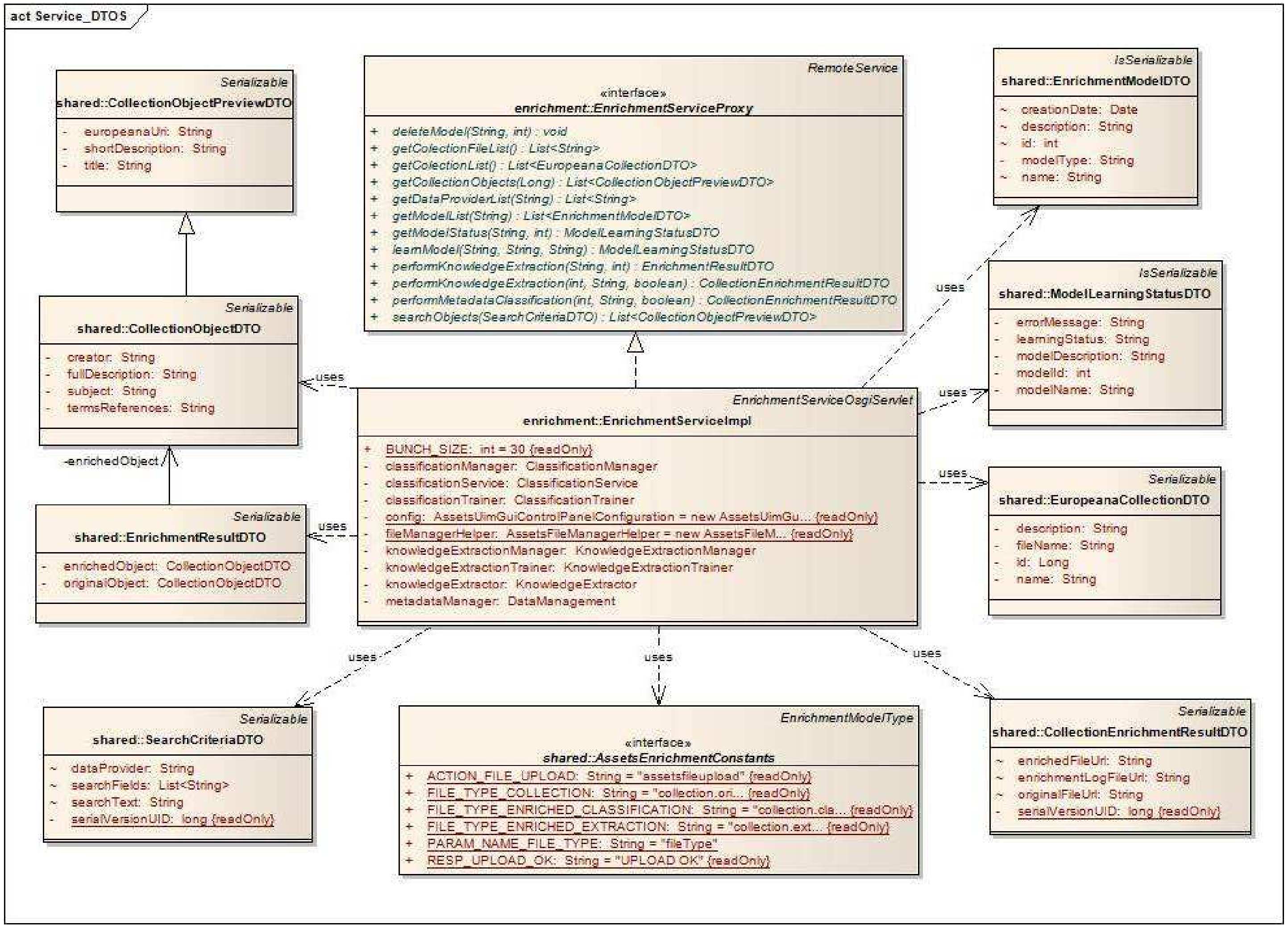

act Service_DTOS
Serializable
shared::CollectionObjectPreviewDTO
- europeanaUri: String
- shortDescription: String
- title: String
Serializable
shared::CollectionObjectDTO
- creator: String
- fullDescription: String
- subject: String
- termsReferences: String
-enrichedObject
Serializable
shared::EnrichmentResultDTO
- enrichedObject: CollectionObjectDTO
- originalObject: CollectionObjectDTO
Serializable
shared::SearchCriteriaDTO
~ dataProvider: String
~ searchFields: List<String>
~ searchText: String
- serialVersionUID: long {readOnly}
RemoteService
«interface»
enrichment::EnrichmentServiceProxy
+ deleteModel(String, int) : void
+ getColectionFileList() : List<String>
+ getColectionList() : List<EuropeanaCollectionDTO>
+ getCollectionObjects(Long) : List<CollectionObjectPreviewDTO>
+ getDataProviderList(String) : List<String>
+ getModelList(String) : List<EnrichmentModelDTO>
+ getModelStatus(String, int) : ModelLearningStatusDTO
+ learnModel(String, String, String) : ModelLearningStatusDTO
+ performKnowledgeExtraction(String, int) : EnrichmentResultDTO
+ performKnowledgeExtraction(int, String, boolean) : CollectionEnrichmentResultDTO
+ performMetadataClassification(int, String, boolean) : CollectionEnrichmentResultDTO
+ searchObjects(SearchCriteriaDTO) : List<CollectionObjectPreviewDTO>
EnrichmentServiceOsgiServlet
enrichment::EnrichmentServiceImpl
+ BUNCH_SIZE: int = 30 {readOnly}
- classificationManager: ClassificationManager
- classificationService: ClassificationService
- classificationTrainer: ClassificationTrainer
- config: AssetsUimGuiControlPanelConfiguration = new AssetsUimGu... {readOnly}
- fileManagerHelper: AssetsFileManagerHelper = new AssetsFileM... {readOnly}
- knowledgeExtractionManager: KnowledgeExtractionManager
- knowledgeExtractionTrainer: KnowledgeExtractionTrainer
- knowledgeExtractor: KnowledgeExtractor
- metadataManager: DataManagement
uses
uses
uses
uses
uses
uses
uses
uses
EnrichmentModelType
«interface»
shared::AssetsEnrichmentConstants
+ ACTION_FILE_UPLOAD: String = "assetsfileupload" {readOnly}
+ FILE_TYPE_COLLECTION: String = "collection.ori... {readOnly}
+ FILE_TYPE_ENRICHED_CLASSIFICATION: String = "collection.cla... {readOnly}
+ FILE_TYPE_ENRICHED_EXTRACTION: String = "collection.ext... {readOnly}
+ PARAM_NAME_FILE_TYPE: String = "fileType"
+ RESP_UPLOAD_OK: String = "UPLOAD OK" {readOnly}
IsSerializable
shared::EnrichmentModelDTO
~ creationDate: Date
~ description: String
~ id: int
- modelType: String
~ name: String
IsSerializable
shared::ModelLearningStatusDTO
- errorMessage: String
- learningStatus: String
- modelDescription: String
- modelId: int
- modelName: String
Serializable
shared::EuropeanaCollectionDTO
- description: String
- fileName: String
- id: Long
- name: String
Serializable
shared::CollectionEnrichmentResultDTO
~ enrichedFileUri: String
~ enrichmentLogFileUri: String
~ originalFileUri: String
- serialVersionUID: long {readOnly}

Figure 14  Enrichment Service API

## 4.3 Software Packaging

The ingestion services are 100% java code, and are developed using Eclipse IDE and Maven built management.

The metadata classification service depends on:

- Jatecs-1.1, for indexing, transformation in vectorial form, learning algorithms, and application of learned model to metadata records.
- Trove-2.1, for the efficient data structures used to store models and vectors.

The knowledge extraction service depends on:

- Stanford-corenlp-11.6.19, for POS tagging, and lemmatization.

The ingestion workflow management web application depends on:

- Ingestion-knowledgeextraction-client, lightweight library used for the remote invocation of knowledge extraction service
- Ingestion-metadataclassification-client, lightweight library used for the remote invocation of metadata classification service
- (ASSETS) Comon-client, lightweight library used for the remote invocation of common functionality of the ASSETS platform. It is used to retrieve the collections and the metadata available on remote ASSETS servers.
- Europeana-uim-gui-controlpanel, the web application implementing the standard functionality of the Europeana ingestion process.

The built process of all the ASSETS components is managed by Hudson, which automatically builds all components every night. The following artefacts are created for the ingestion services:

- ingestion-knowledgeextraction-0.0.1-SNAPSHOT.war, the web application which implements the server side processing of the knowledge extraction service. The service interface is exposed as web services through the REST interface.
- ingestion-metadataclassification-0.0.1-SNAPSHOT.war, the web application which implements the server side processing of the metadata classification service. The service interface is exposed as web services through the REST interface.
- ingestion-knowledgeextraction-client-0.0.1-SNAPSHOT.jar, the lightweight library providing the JAVA API for remote invocation of the knowledge extraction service.
- ingestion-metadataclassification-client-0.0.1-SNAPSHOT.jar, the lightweight library providing the JAVA API for remote invocation of the metadata classification service.
- assets-uim-gui-controlpanel-0.0.1-SNAPSHOT.war, the web application implementing the GUI used for the execution of ingestion and enrichment processes.

## 4.4 Installation and configuration

Both enrichment services require  to be provided the path in the file system where the learned models will be temporarily stored.  The XML files containing batches of metadata records given as input through the training sets are also saved locally. The enriched XML files produced as output are stored permanently in the subfolders of the same path.

For the metadata classification service the properties are set in the file **assets-ingestion-metadataclassification.properties** :

```
path_to_models = ./services/ingestion-metadataclassification/data
path_to_batches = ./services/ingestion-metadataclassification/batches
```

For the knowledge extraction service the properties are set in the file **assets-ingestion-knowledgeextraction.properties**:

```
path_to_models = ./services/ingestion-knowledgeextraction/data
path_to_batches = ./services/ingestion-knowledgeextraction/batches
```

The ingestion workflow management application uses the AssetsIngestionControlPanel.gwt.xml and assets-uim-gui-controlpanel.properties configuration files which are available under project resources. The xml file is used for configuring the GWT engine by defining the *AssetsIngestionControlPanel* module. The information in this file is static and it is used for deploying the GWT application on the server. On the contrary, the .properties file contains information which is specific to each individual server:

```
#folder to upload the training sets for knowledge extraction
knowledge.extraction.models.folder = /assets/enrichment

# folder to upload collections for performing the knowledge
extraction tasks (take care for whitespaces at the end of
properties)
knowledge.extraction.collections.original.folder =
./collections/original
knowledge.extraction.collections.original.baseurl =
http://127.0.0.1:8888/collections/original

# folder to upload collections for performing the knowledge
extraction tasks (take care for whitespaces at the end of
properties)
knowledge.extraction.collections.enriched.folder =
./collections/extraction
knowledge.extraction.collections.enriched.baseurl =
http://127.0.0.1:8888/collections/extraction

#folder to upload the training sets for knowledge extraction (take
care for whitespaces at the end of properties)
classification.models.folder = /assets/enrichment

# folder to upload collections for performing the metadata
classification tasks (take care for whitespaces at the end of
properties)
classification.collections.original.folder = ./collections/original
classification.collections.original.baseurl =
http://127.0.0.1:8888/collections/original

# folder to upload collections for performing the knowledge
extraction tasks (take care for whitespaces at the end of
properties)
classification.collections.enriched.folder =
./collections/classification
classification.collections.enriched.baseurl                         =
http://127.0.0.1:8888/collections/classification
```

# 5. User Manual

This section gives the final user some guidelines to follow in order to define the set of metadata records to be included into the training set, so to ensure an accurate model will be generated by the learning processes. It also describes the XML data format for the specification of training sets for the knowledge extraction and metadata classification services.

## 5.1 Knowledge extraction service

### *5.1.1 Training set definition guidelines*

A training set is composed of a single file that specifies both the set of concepts to be extracted and some annotated training examples where the concept to be extracted is present.

For example, if the set of relevant concepts includes 'Name of person', there must be examples where it is possible to locate the name of a specific concept, e.g., "Thomas Edison invented the filament lamp in America."

Typical relevant concepts are those of Person (e.g., "Thomas Edison", "Barack Obama"), Organisation (e.g., "ONU", "United States of America", "USA"), Location (e.g., "Alps", "Paris"). A relevant concept could be also a specialization of a general one (e.g., Music composer, Non-governmental organisation, Address) or be domain-specific (e.g., Painting technique, Music style, Tool).

The mentions of entities that are instances of such concepts (e.g., "*Oil painting* of a view of the Arno river", "Recording of the *rock* concert held in London in 1982", "Wooden sculpture of a head, sculpted with a c*arving fish tail*") are identified in text and annotated accordingly.

In order to prepare the training set for the Knowledge Extraction service the users should perform the following steps:

1. identify a set of concepts that are relevant for their activities and for which they can provide a critical mass of training examples;
2. identify a set of metadata records to be submitted as training examples. Such metadata records should be representative cases in which a relevant concept (e.g., a person's name) occurs in an unstructured textual field (e.g., in the <description> field);
3. locate the concepts in the textual fields of the examples and verify whether or not it will be necessary to indicate also the exact position of the concept in the text. If the position is not specified, then any instance of the annotated concept in the text is considered as an occurrence of that concept. For example, in "Paris Hilton went to Paris" the annotation of the location "Paris" must specify the position in text[2], otherwise the name of Paris Hilton will also be considered as an occurrence of the location (if Paris Hilton is marked as a person, then the word Paris in Paris Hilton is

2 The various methods to indicate the position will be presented in next section, when discussing an example of a training set.

considered to have two annotations of different types);

4. specify each occurrence of the concept by explicitly specifying the field where the concept occurs and, if needed, its position.

With respect to the number of metadata records to be inserted into the training set, the basic guideline is that the more examples the learning algorithm gets in input, the more probably the learned automatic extractor will be accurate and able to recognize instances of concepts never seen before.

A second guideline is that each concept in the set of relevant concepts should get a relevant number of examples (an indicative number is 1,000 examples per concept). It is not relevant for each metadata record in the training set to contain examples for each possible concept.

### *5.1.2 Training data format*

The training sets created according to the previous guidelines must respect the syntax specified in the XSD schema file *extractionSchema.xsd* that defines the XML elements describing a training set. The *extractionSchema.xsd* file is part of the ASSETS software repository, and it is also fully included in Appendix A. Now we discuss in detail an example of a training set. We will show the steps that a user needs to perform in order to extract knowledge and, in particular, the name of a person from the following metadata record:

```
<europeanaRecord>
        <title>Lamp</title>
        <description>
                Thomas Edison invented the filament lamp in America at almost the
                same time as Joseph Swan did in England. He produced this type of lamp
                in 1880. This particular bulb comes from Pullar's Dye Works in Perth, one
                of the first buildings in Australia to install Edison lights.
        </description>
        <source>Tyne and Wear Imagine</source>
        <provider>CultureGrid ; Uk</provider>
        <identifier>http://www.imagine.org.uk/details/index.php?id=TWCMS:B5141a</i
dentifier>
        <subject> inventors and innovators; people</subject>
        <type>Image</type>
</europeanaRecord>
```

We are interested in extracting the names of persons (*Thomas Edison, Joseph Swan*), the names of places (*America, Australia*), and the names of organization (*Pullar's Dye Works*).

The training set starts with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<extractionTrainingSet xmlns="http://www.example.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/ExtractionSchema.xsd ">
```

The first line declares the XML version used in the file; the second line refers to the XML Schema that defines the syntax for the document, with a reference to the site http://www.example.org, which is only a dummy name that will be replaced with the real URL of the xsd file once the service is deployed.

After the short preamble, the extraction task is defined (XML comments are given inline to better clarify each element purpose):

```
 <extractionTask>
  <!-- This is the name of the field of the records from which information has to be
extracted -->
  <sourceFieldName>description</sourceFieldName>
  <conceptSet>
   <!-- This is a descriptive name for the extraction task -->
   <name>NER for persons, organizations and locations</name>
   <!-- Optional pointer to a resource that describes the concept set -->
   <URI>http://en.wikipedia.org/wiki/Named_entity_recognition</URI>
   <concepts>
    <concept>
     <!-- Name of the concept to be extracted -->
     <name>person</name>
     <!-- Optional pointer to a resource describing the concept -->
     <URI>http://en.wikipedia.org/wiki/Person</URI>
     <!-- Optional name of the target field in the record that has to be filled with
extracted information -->
     <targetField>extractedPerson</targetField>
    </concept>
    <concept>
     <name>organization</name>
     <URI>http://en.wikipedia.org/wiki/Organization</URI>
     <targetField>extractedOrganization</targetField>
    </concept>
    <concept>
     <name>location</name>
     <URI>http://en.wikipedia.org/wiki/Place_(geography)</URI>
     <targetField>extractedLocation</targetField>
    </concept>
   </concepts>
  </conceptSet>
 </extractionTask>
```

The **sourceFieldName** element describes the unstructured textual field of the metadata record from which information has to be extracted, "*description*" in this case.

In this example the set of concepts refer to a Named Entity Recognition (NER) task for persons, organization, and location. After naming the task, the set of concepts is specified: for each concept (described by a **concept** element), the user can optionally provide an URI describing in detail the aim of the task, and the target field of the output XML file where the extracted information will be stored (**targetField**).

It is relevant to note that in the current implementation the optional **targetField** information, if specified, is not used by the service, which instead stores the automatically extracted information in a custom JSON-formatted element, the *dcterms:references* element of the ESE format (which was found to be the appropriate field for storing this type of information). This custom solution is due to the limited possibility of expansion of the ESE format. However the **targetField** field has been left in the training data format in order to support its future use with the EDM format.

Once the concept set has been defined, the user must provide the automatic information extractor with a set of examples in order to train it properly.

All the examples are listed within an **examples** element, and each **example** is composed by two parts, the metadata record and the list of extracted concepts (stand-off annotation):

```
 <examples>
  <example>
   <record>
    <europeanaRecord>
     <title>Lamp</title>
     <description>Thomas Edison invented the filament lamp in America at almost the
same time as Joseph Swan did in England. He produced this type of lamp in 1880. This
particular bulb comes from Pullar's Dye Works in Perth, one of the first buildings in
Australia to install Edison lights.</description>
     <source>Tyne and Wear Imagine</source>
     <provider>CultureGrid ;  Uk</provider>

<identifier>http://www.imagine.org.uk/details/index.php?id=TWCMS:B5141a</identifi
er>
     <subject> inventors and innovators;  people</subject>
     <type>Image</type>
    </europeanaRecord>
   </record>
   <extractedConcept>
    <name>person</name>
    <extractedText>Thomas Edison</extractedText>
    <!-- A position specification is required when multiple instances of the extracted text
appear in the field with different role. In this case no position is required.-->
    <!-- In case a position is necessary, it can be expressed by copying the extracted text
with enough surrounding text in order to make it uniquely identifiable. See examples in
following concept extractions.-->
    <URI>http://viaf.org/viaf/66552944/#Edison, Thomas A. (Thomas Alva), 1847-
1931</URI>
   </extractedConcept>
   <extractedConcept>
    <name>location</name>
    <extractedText>America</extractedText>
    <!-- Also for this case the position is not required. It is just reported as an example. --
>
    <position>
     <context>lamp in America at almost</context>
    </position>
    <URI>http://www.geonames.org/maps/google_39.76_-98.5.html</URI>
   </extractedConcept>
   <extractedConcept>
    <name>person</name>
```

```
        <extractedText>Joseph Swan</extractedText>
        <!-- Position can also be expressed as the offset in number of characters from the
    beginning of the text in the field. -->
        <position>
          <startCharacterPosition>80</startCharacterPosition>
          <endCharacterPosition>91</endCharacterPosition>
        </position>
        <URI>http://viaf.org/viaf/15100261/#Swan, Joseph Wilson, 1828-1914</URI>
      </extractedConcept>
      <extractedConcept>
        <name>location</name>
        <extractedText>England</extractedText>
        <URI>http://www.geonames.org/2635167/united-kingdom-of-great-britain-and-
    northern-ireland.html</URI>
      </extractedConcept>
      <extractedConcept>
        <name>organization</name>
        <extractedText>Pullar's Dye Works</extractedText>

    <URI>http://canmore.rcahms.gov.uk/en/site/127331/details/perth+pullar+s+dyeworks/
    </URI>
      </extractedConcept>
      <extractedConcept>
        <name>location</name>
        <extractedText>Perth</extractedText>
        <URI>http://www.geonames.org/2063523/perth.html</URI>
      </extractedConcept>
      <extractedConcept>
        <name>location</name>
        <extractedText>Australia</extractedText>
        <URI>http://www.geonames.org/2077456/commonwealth-of-australia.html</URI>
      </extractedConcept>
      <extractedConcept>
        <name>person</name>
        <extractedText>Edison</extractedText>
        <position>
          <context>to install Edison lights</context>
        </position>
        <URI>http://viaf.org/viaf/66552944/#Edison, Thomas A. (Thomas Alva), 1847-
    1931</URI>
      </extractedConcept>
     </example>
    </examples>
```

For example, for the name of person “Thomas Edison” the user should prepare the XML element:

```
      <extractedConcept>
        <name>person</name>
        <extractedText>Thomas Edison</extractedText>
        <URI>http://viaf.org/viaf/66552944/#Edison, Thomas A. (Thomas Alva), 1847-
    1931</URI>
      </extractedConcept>
```

where there is no position indicated since there are no ambiguities in the description field of the metadata record this **extractedConcept** refers to.

In the case the position must be indicated, the user may choose among different alternatives. They may select the context where the instance occurs, like, e.g., for the second concept to be extracted:

```
<position>
  <context>lamp in America at almost</context>
</position>
```

Alternatively, they may indicate the position in the description where the instance occurs. An example of this second option can be found in the third extracted concept:

```
<position>
  <startCharacterPosition>80</startCharacterPosition>
  <endCharacterPosition>91</endCharacterPosition>
</position>
```

Appendix A contains the complete listing of the above example.

### *5.1.3 Stand-alone test user interface*

Though the knowledge extraction service is meant to be accessed by users as a plugin of the ingestion workflow, we have developed a graphical user interface (GUI) that allows the user to directly connect to the service. The GUI uses the knowledge extraction client library to access the server through the REST interface that the server-side application exposes.

The original purpose of the test GUI is to allow developers to have a direct access to the knowledge extraction service for testing and debug purposes, but also skilled users can benefit from its availability.

In the knowledge extraction client repository, the GUI application is defined in the TestIngestionKnowledgeExtractionGui class, which is part of the eu.europeana.assets.ingestion.knowledgeextraction.client package.

The GUI is composed by three areas (see Figure 15):

- the top left area is devoted to set and start training requests;
- the bottom left area is devoted to set and start enrichment requests;
- the right area gives feedback on any request.

In order to train a new knowledge extraction model, the user can select a training file (Figure 16) and launch the learning process. During the training process the GUI gives periodical feedback on the status.

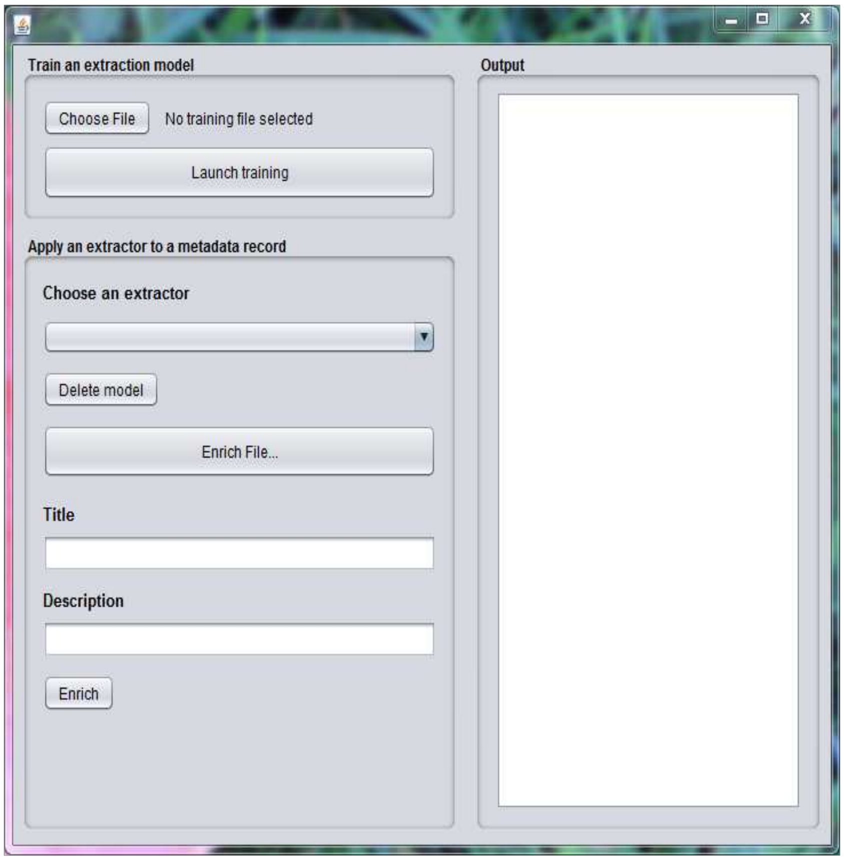
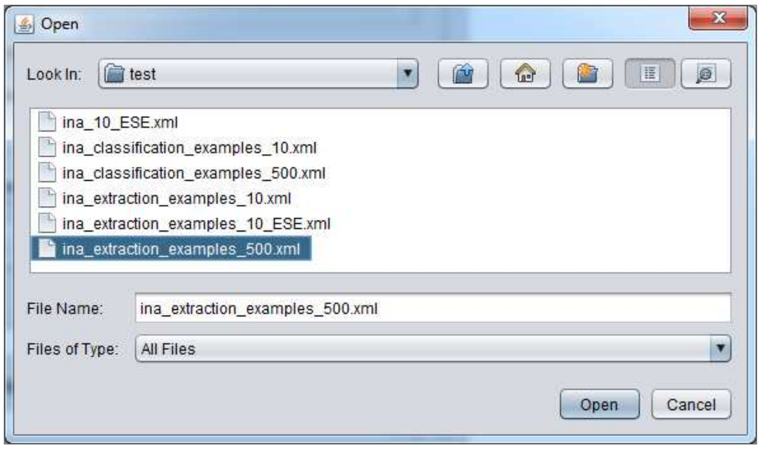


*Figure 15  Test GUI for knowledge extraction*

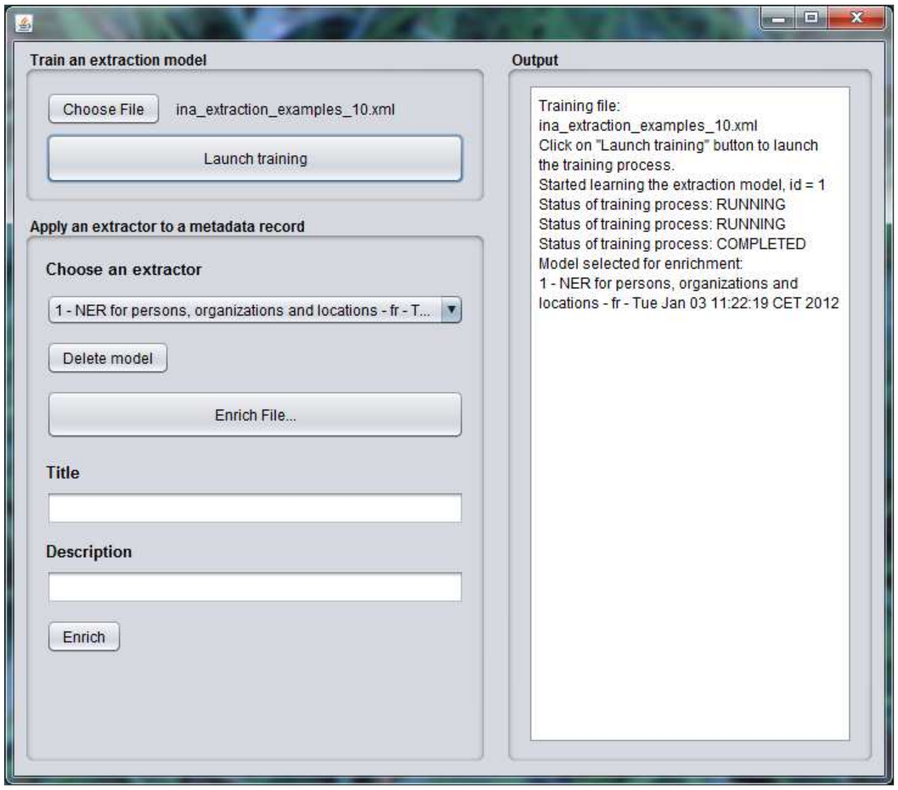
Once completed, the new knowledge extraction model is made available for selection in the list of models (Figure 17).



ina_10_ESE.xml
ina_classification_examples_10.xml
ina_classification_examples_500.xml
ina_extraction_examples_10.xml
ina_extraction_examples_10_ESE.xml
ina_extraction_examples_500.xml

File Name: ina_extraction_examples_500.xml
Files of Type: All Files

*Figure 16 Selection of training file*

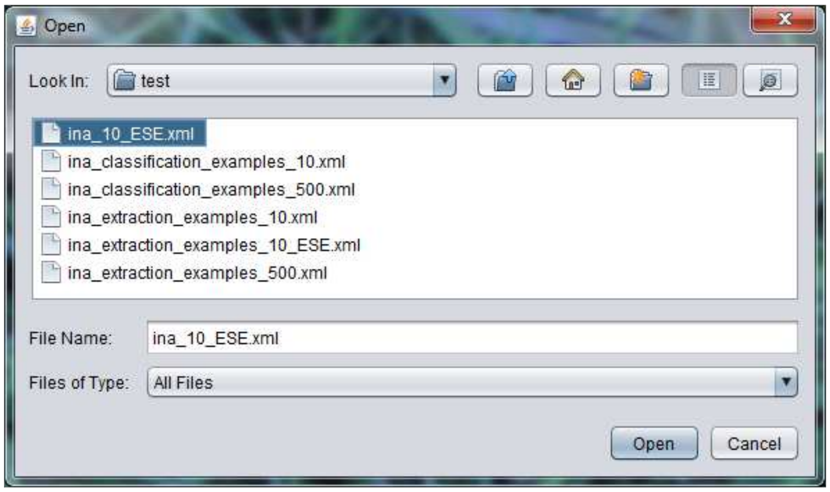


ina_extraction_examples_10.xml

1 - NER for persons, organizations and locations - fr - T...

```
Training file:
ina_extraction_examples_10.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - NER for persons, organizations and
locations - fr - Tue Jan 03 11:22:19 CET 2012
```

*Figure 17 Output of training process and selection of the trained model for enrichment of metadata records*



ina_10_ESE.xml
ina_classification_examples_10.xml
ina_classification_examples_500.xml
ina_extraction_examples_10.xml
ina_extraction_examples_10_ESE.xml
ina_extraction_examples_500.xml

File Name: ina_10_ESE.xml

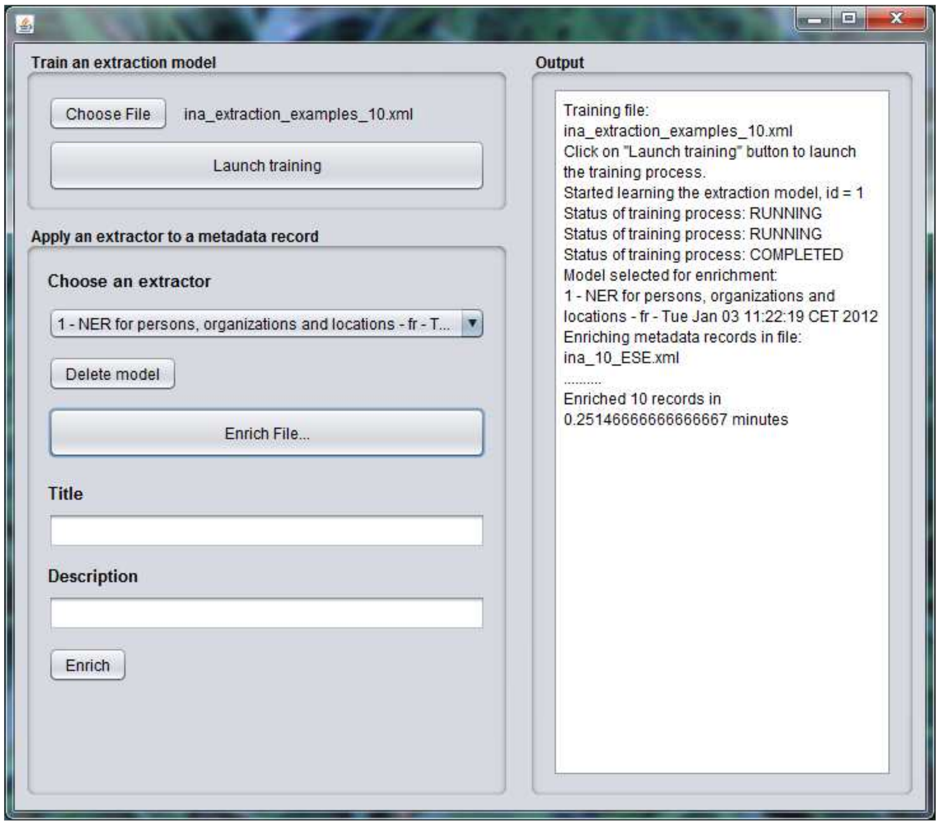
*Figure 18 Selection of an XML file for enrichment*

A drop-down list displays all the available knowledge extraction models. The user can select an entire XML file containing metadata records in ESE format for batch processing (Figure 18), or can specify the values of the title and description fields of a "dummy" custom-made record, in order to quickly check the output generated by a model.

When processing a file, the output window gives feedback on the progress of the process (Figure 19).

The enriched metadata records are saved in the same directory of the source file, with the "enrichedExtraction." prefix.

In the example, the output of the enrichment process of the "ina_10_ESE.xml" file is saved in the "enrichedExtraction.ina_10_ESE.xml". Since the input and the output files differ only for the additional information added by the knowledge extraction process, the output file can be used as input for another enrichment process or as a direct substitute of the input file in the ingestion process.


Train an extraction model
Choose File
ina_extraction_examples_10.xml
Launch training
Apply an extractor to a metadata record
Choose an extractor
1 - NER for persons, organizations and locations - fr - T...
Delete model
Enrich File...
Title
Description
Enrich
Output

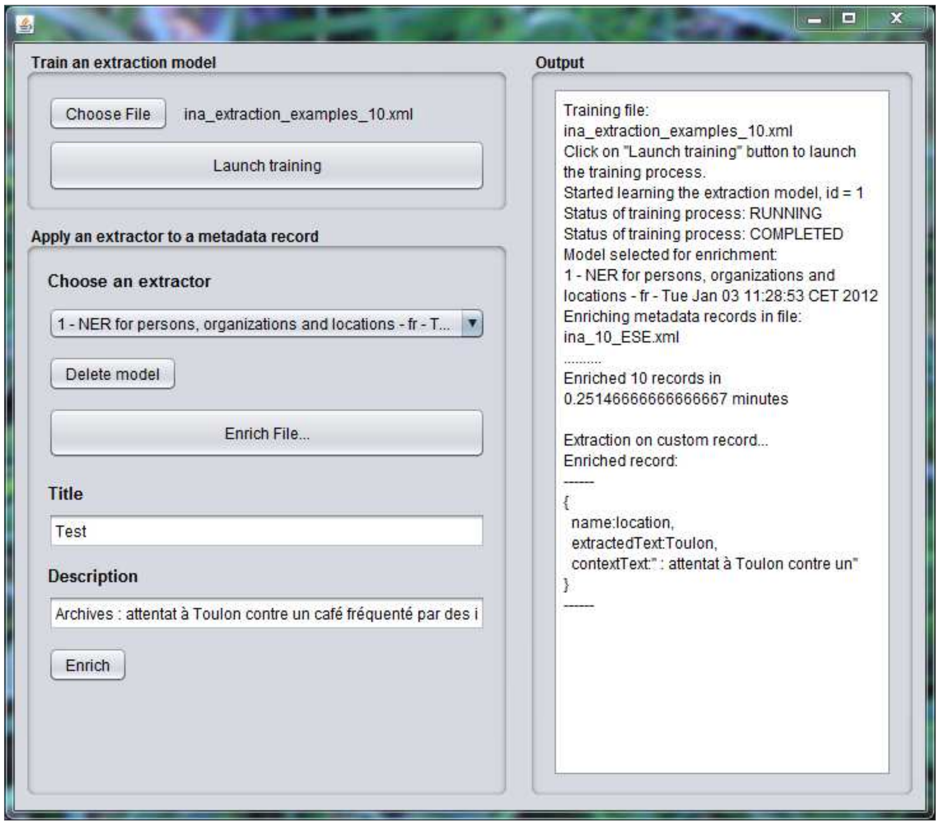
```
Training file:
ina_extraction_examples_10.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - NER for persons, organizations and
locations - fr - Tue Jan 03 11:22:19 CET 2012
Enriching metadata records in file:
ina_10_ESE.xml
..........
Enriched 10 records in
0.25146666666666667 minutes
```

*Figure 19 Enrichment of an XML file*

When applied to a custom-made metadata record, the output window shows a list of the concepts extracted from the metadata record (Figure 20).

It is worth to be noted the "- fr -" suffix to the names of available extractors is automatically determined by a language recognition module that is part of the knowledge extraction service.

A user can simultaneously send more than one train and/or enrichment requests, since the service is designed to support concurrent requests. The only issue is that the feedback from the various requests will be mixed when printed in the output window.


Train an extraction model
Choose File
ina_extraction_examples_10.xml
Launch training
Apply an extractor to a metadata record
Choose an extractor
1 - NER for persons, organizations and locations - fr - T...
Delete model
Enrich File...
Title
Test
Description
Archives : attentat à Toulon contre un café fréquenté par des i
Enrich
Output

```
Training file:
ina_extraction_examples_10.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - NER for persons, organizations and
locations - fr - Tue Jan 03 11:28:53 CET 2012
Enriching metadata records in file:
ina_10_ESE.xml
..........
Enriched 10 records in
0.25146666666666667 minutes

Extraction on custom record...
Enriched record:
------
{
  name:location,
  extractedText:Toulon,
  contextText:" : attentat à Toulon contre un"
}
------
```

*Figure 20  Enrichment of a custom-made metadata record*

## 5.2 Metadata classification service

### *5.2.1 Training set definition guidelines*

The user needs to prepare an XML file containing both the taxonomy of semantic categories its metadata belong to and some training examples composed of metadata record and desired category related to the chosen taxonomy. For instance, if a user has the following metadata record:

```
<recording>
  <title>The Marriage of Figaro</title>
  <author>Wolfgang Amadeus Mozart</author>
  <year>1943</year>
  <director>Paul Breisach</director>
  <orchestra>Metropolitan Opera Orchestra</orchestra>
  <location>New York</location>
</recording>
```

and wants it to be classified into the “Classical” music category, then the user needs to prepare a labelled training example where the category “Classical” is associated to the previous metadata record.

Concerning the taxonomy, users with similar thematic collections are invited to select a common classification taxonomy. In fact, if there were a single taxonomy, the classifier induced by a training set coming from a given user might also be used with success on metadata coming from another user.

In order to prepare a training set for the metadata classification service, the user should perform the following steps:

1. select a classification taxonomy. The service is able to work with both a flat or a hierarchical taxonomy;
2. select a set of metadata records;
3. associate to each metadata record (selected in the previous step) one or more categories belonging to the taxonomy defined at step 1.

With respect to the number of metadata record to be inserted into the training set, the basic guideline is that the more examples the learning algorithm gets in input, the more probably the learned automatic classifier will be accurate in the assignment of the categories.

A second guideline is that each relevant category in the taxonomy should get a relevant number of examples (an indicative number is having a training set of at least 1,000 metadata records, with at least 10 examples for the least popular category).

It is relevant to have examples for each relevant category in the taxonomy, because any category that is not represented by at least one example will be discarded during the learning process and never assigned during the automatic classification process.

### *5.2.2 Training data format*

The training sets created according to the previous guidelines must respect the syntax specified in the XSD schema file *classificationSchema.xsd* that defines the XML elements describing a training set. *The classificationSchema.xsd* file is part of the ASSETS software repository, and it is also fully included in Appendix B.

Now we discuss an example of a training set in detail. We will show the steps that a user needs to perform in order to define a training set for the classification of music recording by genre.

As for the previous service, the training set starts with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<classificationTrainingSet xmlns="http://www.example.org/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/ClassificationSchema.xsd ">
```

Then, a section defines the classification taxonomy and its properties (like in previous section in-line comments describe the meaning of each XML element):

```
 <classificationSchema>
  <!-- A meaningful name for the classification schema -->
  <name>a simple music genre classificiation schema</name>
  <!-- Optional URI pointing to a description of the classification schema -->
  <URI>http://en.wikipedia.org/wiki/Music_genre</URI>
  <!-- Type of classification schema
       multiLabel: each record can be assigned to zero, one, or more than one class
       singleLabel: each record has to be assigned to one and only one class
  -->
  <type>singleLabel</type>
  <!-- List of classes -->
  <classes>
    <!-- Definition of a class -->
    <class>
      <!-- Class name -->
      <name>classical</name>
      <!-- Optional URI describing the class -->
      <URI>http://en.wikipedia.org/wiki/Classical_music</URI>
    </class>
    <class>
      <name>baroque</name>
      <URI>http://en.wikipedia.org/wiki/Baroque_music</URI>
      <!-- The classification schema could be hierachical. In this case "baroque" is a more
specific definition of a classical music genre. -->
      <!-- If a record is assigned to baroque it is implicitly also an example of classical
music. -->
      <!-- If a record is assigned to classical, it means that, though the record refers to
classical music, it does not belong to any of the more specific classes. -->
      <parentClassName>classical</parentClassName>
    </class>
    <class>
      <name>modern</name>
      <URI>http://en.wikipedia.org/wiki/20th-century_classical_music</URI>
      <parentClassName>classical</parentClassName>
    </class>
    <class>
      <name>romantic</name>
      <URI>http://en.wikipedia.org/wiki/Romantic_music</URI>
      <parentClassName>classical</parentClassName>
    </class>
    <class>
      <name>jazz</name>
      <URI>http://en.wikipedia.org/wiki/Jazz</URI>
    </class>
    <class>
      <name>bebop</name>
      <URI>http://en.wikipedia.org/wiki/Bebop</URI>
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>funky</name>
      <URI>http://en.wikipedia.org/wiki/Funk</URI>
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>swing</name>
      <URI>http://en.wikipedia.org/wiki/Swing_music</URI>
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>popular</name>
      <URI>http://en.wikipedia.org/wiki/Popular_music</URI>
    </class>
    <class>
      <name>country</name>
      <URI>http://en.wikipedia.org/wiki/Country_music</URI>
      <parentClassName>popular</parentClassName>
    </class>
    <class>
      <name>punk</name>
      <URI>http://en.wikipedia.org/wiki/Punk_rock</URI>
      <parentClassName>popular</parentClassName>
    </class>
    <class>
      <name>rap</name>
      <URI>http://en.wikipedia.org/wiki/Hip_hop_music</URI>
      <parentClassName>popular</parentClassName>
    </class>
  </classes>
</classificationSchema>
```

The two relevant properties of a **classificationSchema** are

(I) the specification of the single- or multi-label classification model,

(II) the specification of any eventual hierarchy relation among categories.

In the example above, the classification schema is said to be *single-label*, i.e., one and only one category can be assigned to every metadata record. Then some categories are said to be children of other categories, e.g., *baroque* is a child of *classic.* The learning algorithm exploits hierarchical information in order to perform a more efficient learning process and also to improve the accuracy of the learned classification model.

The rest of the training file is composed of the **examples list**, i.e., manually classified metadata records. Each **example** is composed of the original metadata record and a list of **assignedClass** elements, listing the names of the classes assigned to the record.

```
<examples>
 <example>
  <record>
   <recording>
    <title>The Marriage of Figaro</title>
    <author>Wolfgang Amadeus Mozart</author>
    <year>1943</year>
```

```
        <director>Paul Breisach</director>
        <orchestra>Metropolitan Opera Orchestra</orchestra>
        <location>New York</location>
      </recording>
    </record>
    <assignedClass>classical</assignedClass>
  </example>
  <example>
    <record>
      <recording>
        <title> Brandenburg Concerto No. 1 - 1</title>
        <author>Johann Sebastian Bach</author>
        <orchestra>The Busch Chamber Players</orchestra>
        <URI>http://en.wikipedia.org/wiki/File:Bach_-_Brandenburg_Concerto_No._1_-
_1._Allegro.ogg</URI>
      </recording>
    </record>
    <assignedClass>baroque</assignedClass>
  </example>
  <example>
    <record>
      <recording>
        <title>Für Elise</title>
        <title>Bagatelle No. 25 in A minor</title>
        <author>Ludwig van Beethoven</author>
        <year>1867</year>
      </recording>
    </record>
    <assignedClass>romantic</assignedClass>
  </example>
  <example>
    <record>
      <recording>
        <title>Salt Peanuts</title>
        <composed>1942</composed>
        <year>1947</year>
        <author>Dizzy Gillespie</author>
        <description>Dizzy played for Lucky Millinder's band in the early '40s. It was a riff
this band played, after a Dizzy solo in the tune "Little John Special", that Dizzy developed
into his tune "Salt Peanuts".</description>
        <URI>http://www.youtube.com/watch?v=kOmA8LOw258</URI>
      </recording>
    </record>
    <assignedClass>bebop</assignedClass>
  </example>
  <example>
    <record>
      <recording>
```

```
        <title>Gotta Lotta Love</title>
        <author>Tracy Marrow</author>
        <author>Ice-T</author>
        <album>Home invasion</album>
        <year>1993</year>
      </recording>
    </record>
    <assignedClass>rap</assignedClass>
  </example>
  <example>
    <record>
      <recording>
        <title>The river unbroken</title>
        <author>Dolly Rebecca Parton</author>
        <year>1987</year>
        <album>Rainbox</album>
        <label>CBS</label>
        <producer>Steve "Gold-E" Goldstein</producer>
      </recording>
    </record>
    <assignedClass>country</assignedClass>
  </example>
</examples>
```

Appendix B contains the complete listing of the above example.

In the example, the metadata records in the training set are in a proprietary format, while the latest release of the metadata classification service expects them to be usually in ESE format.

The output of the metadata classification process is an exact copy of the metadata record given in input with additional **dc:subject** fields, one for each assigned class. The name of the assigned class is prefixed with the string “assets:” in order to distinguish automatically assigned classes from the original values of that field, e.g.:

```
<record>
  <dc:identifier>0023420002</dc:identifier>
  <dcterms:isPartOf>http://www.ina.fr/</dcterms:isPartOf>
  <dc:title>Le bar du bébète show : [émission du 20 février 1995]</dc:title>
  <dcterms:issued>1995-02-20</dcterms:issued>
  <dcterms:extent>0h4m59s</dcterms:extent>
  <dc:description xml:lang="fr">L'actualité de la semaine parodiée par les
marionnettes du Bébête show. Cette semaine :
- Au bar : François MITTERRAND, Henri EMMANUELLI, Jack LANG et Lionel JOSPIN
candidat à l'élection présidentielle. Arrivée de Philippe SEGUIN qui a "affronté" Edouard
BALLADUR à l'émission 7/7. Ce dernier entraine JOSPIN à être agressif.
- Publicité : "Ecoutes express" par les productions Charles Pasqua.
- Au bar : Nicolas SARKOZY et Jacques CHIRAC qui se moque d'Edouard BALLADUR.
- Bande annonce : "Chirac était son nom" avec Jacques CHIRAC en Jésus et Edouard
BALLADUR en Judas.
```

```
</dc:description>
    <dc:subject>Humour</dc:subject>
    <dc:type>Animation</dc:type>
    <dc:type>Sketch</dc:type>
    <dc:subject>François Mitterrand</dc:subject>
    <dc:subject>Henri Emmanuelli</dc:subject>
    <dc:subject>Jack Lang</dc:subject>
    <dc:subject>Lionel Jospin</dc:subject>
    <dc:subject>Philippe Seguin</dc:subject>
    <dc:subject>Charles Pasqua</dc:subject>
    <dc:subject>Edouard Balladur</dc:subject>
    <dc:contributor>Guy Lecluyse</dc:contributor>
    <dc:contributor>Christian Briand</dc:contributor>
    <dc:contributor>Eric Gindre</dc:contributor>
    <dc:contributor>Michel Guidoni</dc:contributor>
    <dc:contributor>Jean Claude Poirot</dc:contributor>
    <dc:publisher>TF1(diffuseur)</dc:publisher>
    <dcterms:medium>video/mpeg</dcterms:medium>
    <dc:language>fr</dc:language>
    <dc:rights>Institut National de l'Audiovisuel</dc:rights>

<europeana:object>http://www.ina.fr/images_v2/320x240/0023420002.jpeg</europe
ana:object>
    <europeana:provider>ASSETS</europeana:provider>
    <europeana:type>VIDEO</europeana:type>
    <europeana:dataProvider>Institut National de
l'Audiovisuel</europeana:dataProvider>
    <europeana:isShownAt>http://www.ina.fr/video/0023420002/le-bar-du-bebete-
show-emission-du-20-fevrier-1995.fr.html#xtor=AL-3</europeana:isShownAt>
    <dc:subject>assets:Humour</dc:subject>
    <dc:subject>assets:Information politique économique sociale</dc:subject>
  </record>
```

The above metadata record of a video recording uses the dc:subject field to store different types of information, from a thematic class (Humour) to the names of people appearing in the video. The automatically assigned classes are listed at the end of the record.

### *5.2.3 Stand-alone test user interface*

Though the metadata classification service is meant to be accessed by users as a plugin of the ingestion workflow, we have developed a graphical user interface (GUI) that allows the user to directly connect to the service. The GUI uses the metadata classification client library to access the server through the REST interface that the server-side application exposes.

The original purpose of the test GUI is to allow developers to have a direct access to the metadata classification service for testing and debug purposes.

In the metadata classification client repository, the GUI application is defined in the TestIngestionMetadataClassificationGui class, which is part of the eu.europeana.assets.ingestion.metadataclassification.client package.

The GUI is composed of three areas (Figure 21):

- the top left area is devoted to set and start training requests;
- the bottom left area is devoted to set and start enrichment requests;
- the right area gives feedback on any request.

In order to train a new metadata classification model, the user can select a training file (Figure 22) and launch a training. During the training process, the GUI gives periodical feedback on the status of the training process.

Once completed, the new metadata classification model is made available for selection in the list of models (Figure 23).

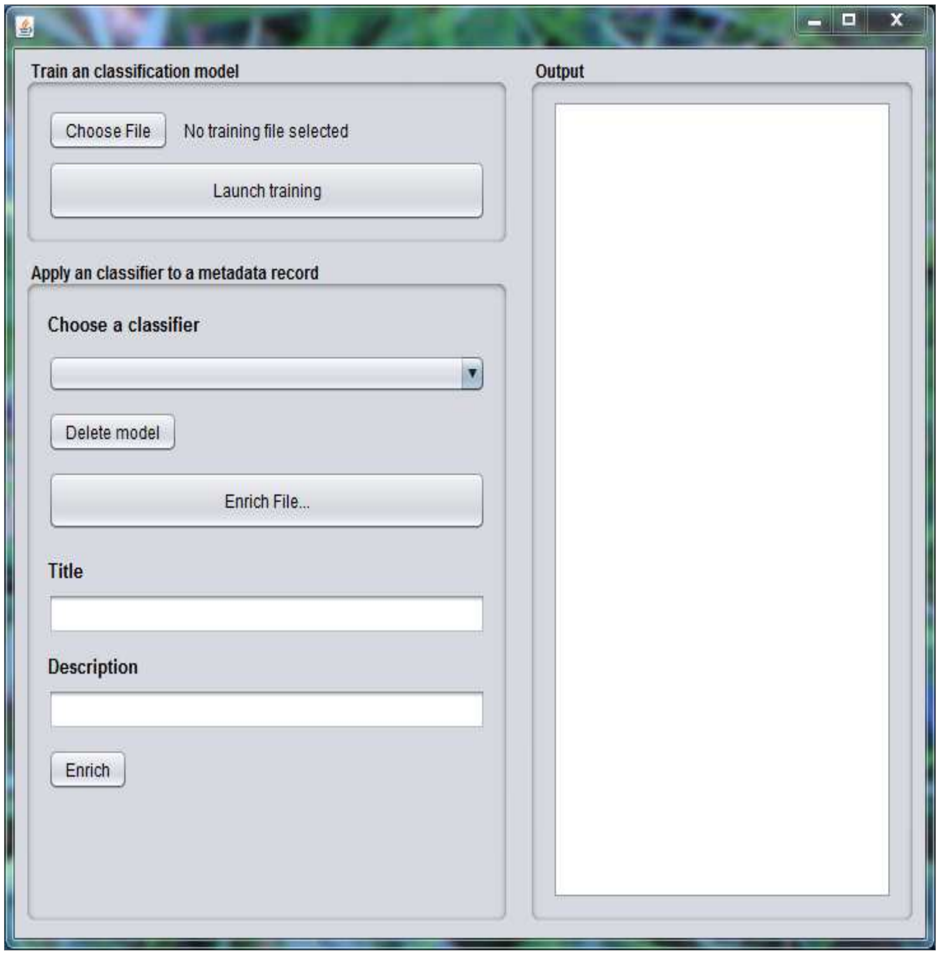
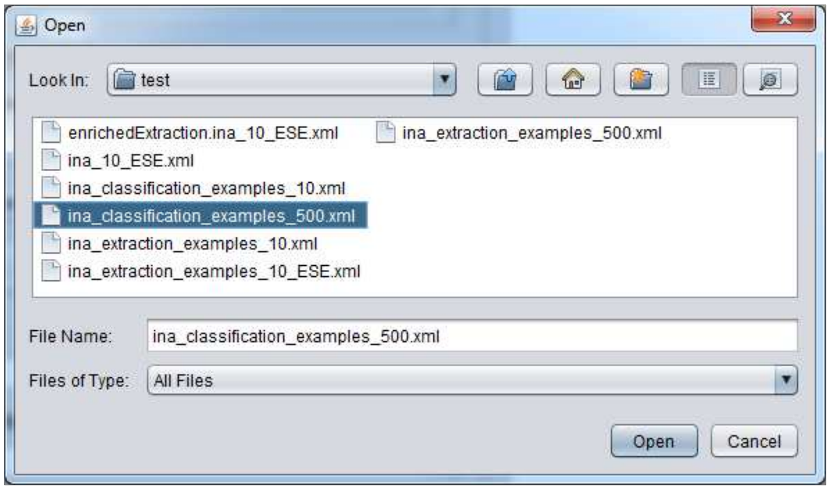


*Figure 21 Test GUI for metadata classification*

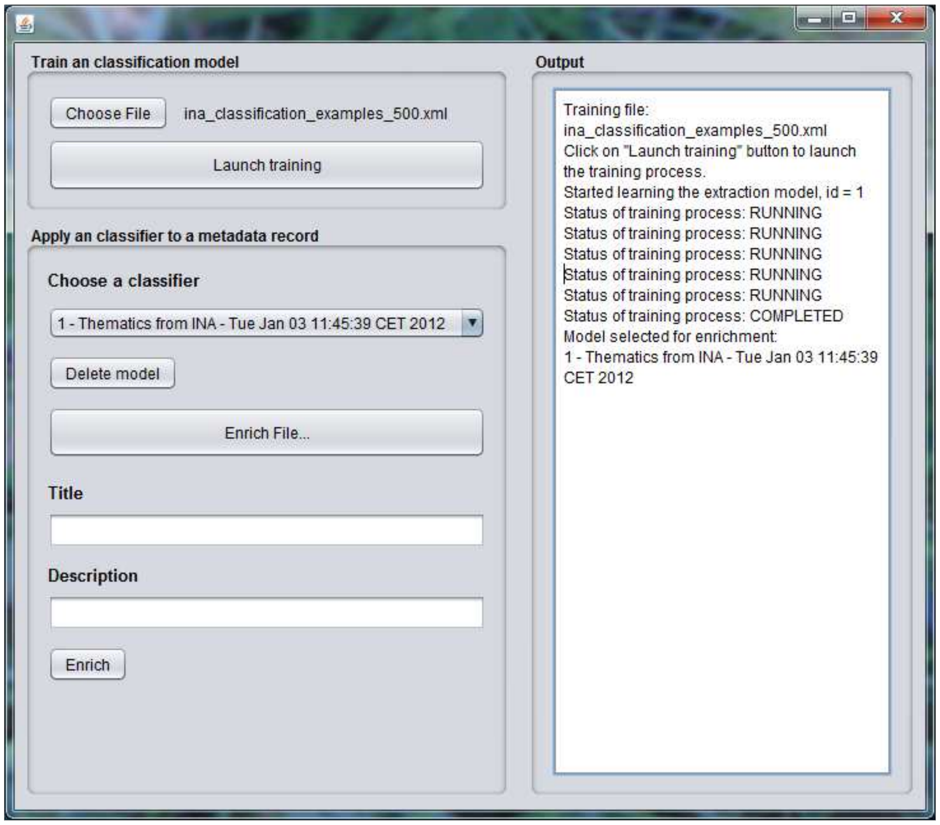


enrichedExtraction.ina_10_ESE.xml
ina_extraction_examples_500.xml
ina_10_ESE.xml
ina_classification_examples_10.xml
ina_classification_examples_500.xml
ina_extraction_examples_10.xml
ina_extraction_examples_10_ESE.xml

File Name: ina_classification_examples_500.xml

Files of Type: All Files

*Figure 22  Selection of training file*



ina_classification_examples_500.xml

1 - Thematics from INA - Tue Jan 03 11:45:39 CET 2012

```
Training file:
ina_classification_examples_500.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - Thematics from INA - Tue Jan 03 11:45:39
CET 2012
```

*Figure 23  Output of training process and selection of the trained model for enrichment of metadata records*

A drop-down list displays all the available metadata classification models. The user can select an entire XML file containing metadata records in ESE format for batch processing (Figure 24), or can specify the values of the title and description fields of a "dummy" custom-made record, in order to quickly check the output generated by a model.

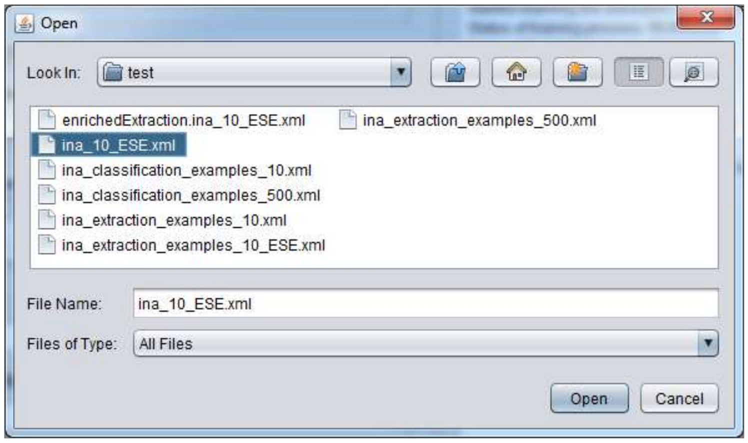
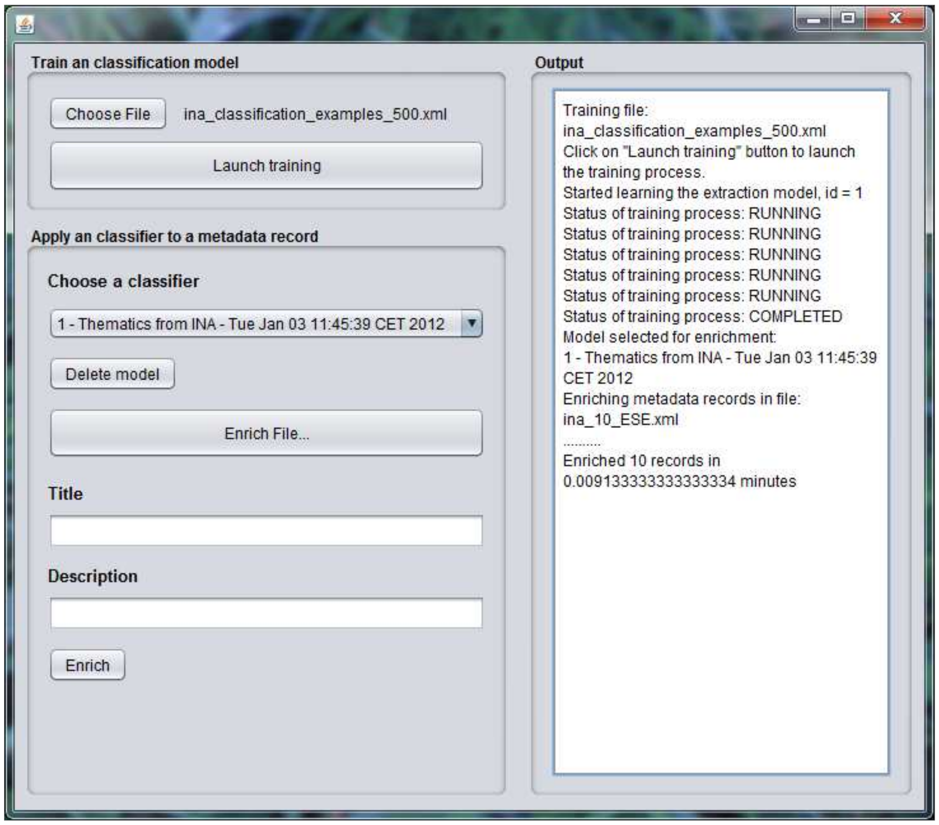

Open
Look In:
test
enrichedExtraction.ina_10_ESE.xml
ina_extraction_examples_500.xml
ina_10_ESE.xml
ina_classification_examples_10.xml
ina_classification_examples_500.xml
ina_extraction_examples_10.xml
ina_extraction_examples_10_ESE.xml
File Name:
ina_10_ESE.xml
Files of Type:
All Files
Open
Cancel

*Figure 24  Selection of an XML file for enrichment*

When processing a file, the output window gives feedback on the progress of the process (Figure 25).

The enriched metadata records are saved in the same directory of the source file, with a “enrichedClassification.” prefix. In the example, the output of the enrichment process of the “ina_10_ESE.xml” file is saved in the “enrichedClassification .ina_10_ESE.xml”. Since the input and the output files differ only for the additional information added by the metadata classification process, the output file can be used as input to another enrichment process, or as a direct substitute of the input file in the ingestion process.

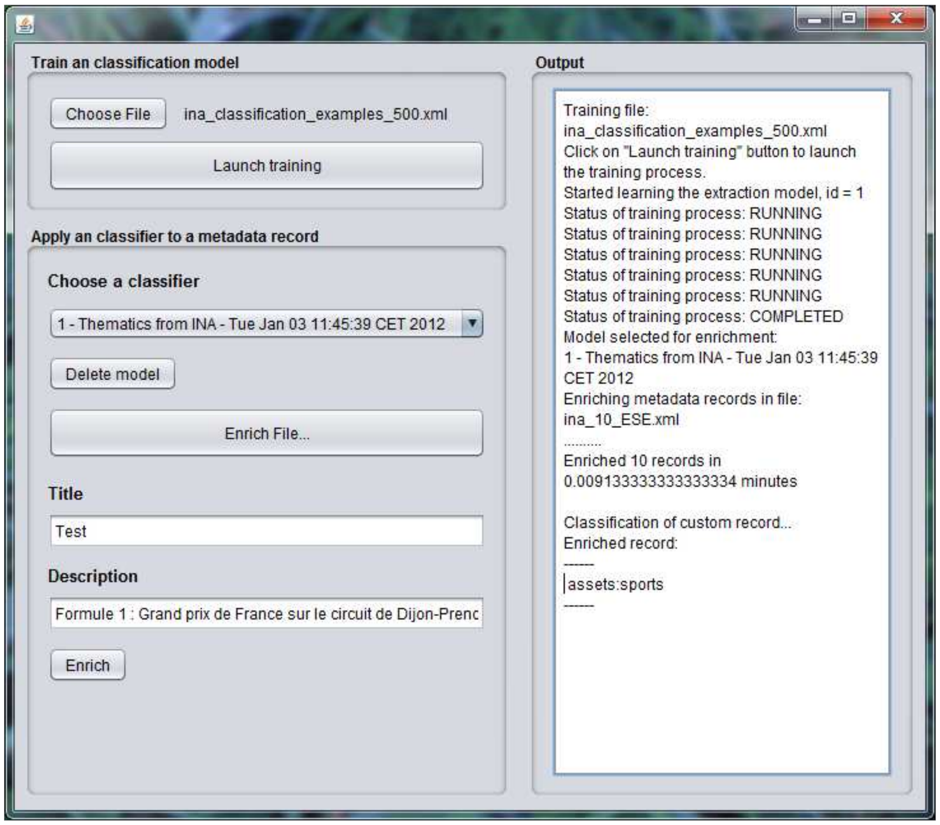

Train an classification model
Choose File
ina_classification_examples_500.xml
Launch training
Apply an classifier to a metadata record
Choose a classifier
1 - Thematics from INA - Tue Jan 03 11:45:39 CET 2012
Delete model
Enrich File...
Title
Description
Enrich
Output

```
Training file:
ina_classification_examples_500.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - Thematics from INA - Tue Jan 03 11:45:39
CET 2012
Enriching metadata records in file:
ina_10_ESE.xml
..........
Enriched 10 records in
0.009133333333333334 minutes
```

*Figure 25  Enrichment of an XML file*

When applied to a custom-made metadata record, the output window shows a list of the classes assigned to the metadata record (Figure 26).

A user can simultaneously send more than one train and/or enrichment requests, since the service is designed to support concurrent requests. The only issue is that the feedback from the various requests will be mixed when printed in the output window.


Train an classification model
Choose File
ina_classification_examples_500.xml
Launch training
Apply an classifier to a metadata record
Choose a classifier
1 - Thematics from INA - Tue Jan 03 11:45:39 CET 2012
Delete model
Enrich File...
Title
Test
Description
Formule 1 : Grand prix de France sur le circuit de Dijon-Prenc
Enrich
Output

```
Training file:
ina_classification_examples_500.xml
Click on "Launch training" button to launch
the training process.
Started learning the extraction model, id = 1
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: RUNNING
Status of training process: COMPLETED
Model selected for enrichment:
1 - Thematics from INA - Tue Jan 03 11:45:39
CET 2012
Enriching metadata records in file:
ina_10_ESE.xml
..........
Enriched 10 records in
0.009133333333333334 minutes

Classification of custom record...
Enriched record:
------
assets:sports
------
```

*Figure 26  Enrichment of a custom-made metadata record*

## 5.3 Ingestion Workflow GUI

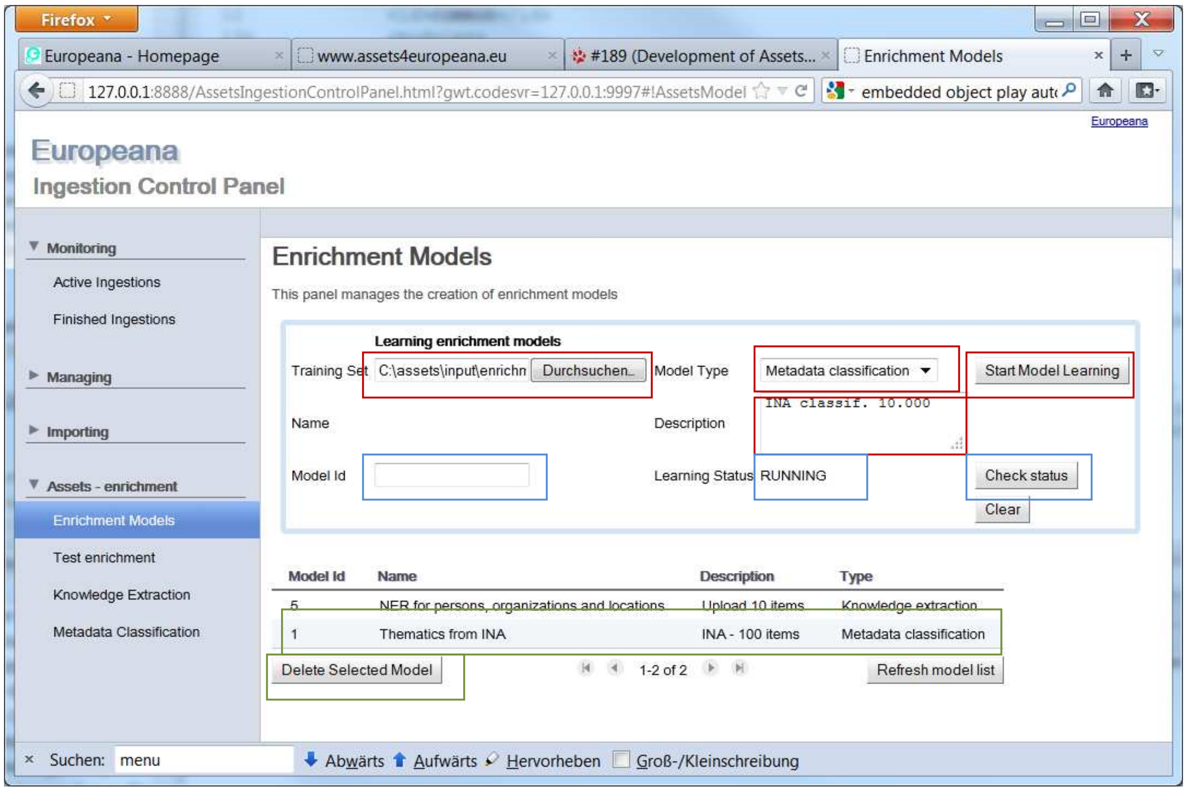
The GUI for the ingestion workflow management service is implemented as an extension of the UIM- Controlpanel. This provides a user friendly interface for management of enrichment related activities, which are:

- training (learning) of enrichment models,
- testing the enrichment models
- execution of knowledge extraction tasks,
- execution of metadata classification tasks.

In the following chapters, we present some screenshots that show how the functionalities listed above have been implemented and the control elements available for user interaction.

### *5.3.1 Enrichment model learning panel*

The panel used for creation and management of the enrichment models offers the possibility to run the following scenarios: create new models, visualize the existing models, verify the learning status and delete an existing model (See Figure 27)



Enrichment Models

This panel manages the creation of enrichment models

Learning enrichment models

Training Set: C:\assets\input\enrichn

Model Type: Metadata classification

Name:

Description: INA classif. 10.000

Model Id:

Learning Status: RUNNING

| Model Id | Name | Description | Type |
|---|---|---|---|
| 5 | NER for persons, organizations and locations | Upload 10 items | Knowledge extraction |
| 1 | Thematics from INA | INA - 100 items | Metadata classification |

Suchen: menu

*Figure 27 Enrichment model learning screen*

To create a **new enrichment model,** the user must provide a file containing the training set defined according to the requirements specified in Sections 5.2.1 and 5.1.1, respectively. The input fields required for this operation (training set file, model type, description) are marked with red colour in Figure 27. By pressing the “Start Model Learning” button, the training file will be submitted to the server and the process of learning the enrichment model will be started. The status of the model learning will be displayed in the corresponding field. The model learning operation could take some time, depending on the size of the training set. During the learning operation, the panel will display the status RUNNING for the current model. The client is not notified automatically about the finalization of the learning operation, but the user is able to check if the new model has been created by **refreshing the model list** (i.e. press the “Refresh model list” button).

The **status of the existing models** can be verified by providing the id of the mode in the Model Id field and pressing the “Check Status” button. These controls are marked with the blue colour in Figure 27. The model Id is displayed in the first column of the model list table. The same functionality can be used to verify if the model learning was completed or not.

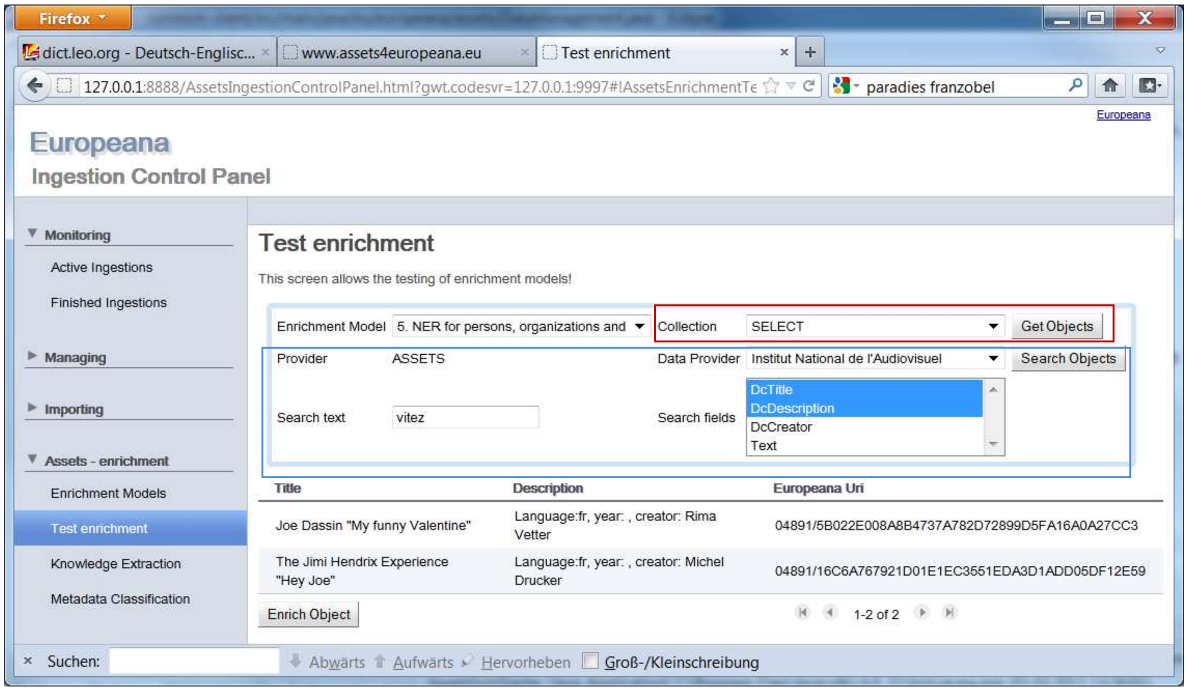
At the bottom of the screen, the list of the existing models is displayed. The **deletion of an existing model** can be performed by selecting the required model within the model list table and pressing the “Delete Selected Model” button (marked with green colour).

### 5.3.2 *Test enrichment panel*

The enrichment models created at the end of the process described in the previous section can be used to enrich metadata collections available in ESE format. Anyway, this is a time consuming activity and the librarians using these services would need to test them before performing the enrichment of large collections. They might also need to test if the system was able to learn the models for the given training set. The test enrichment panel was created especially for supporting this kind of tasks (see Figure 28). It allows the user to select a model and to use it for improving the description of a selected item. Differently from the collection enrichment panels (knowledge extraction, metadata classification), this panel retrieves the objects from the ASSETS servers and not from a metadata file (because the invocation of the search object functionality is required to support this task).

The process of testing the enrichment consists of 3 steps:

- **Select enrichment model**. The enrichment model combo box displays all enrichment models available in the system, and the user has to select one model for testing it. See Section 5.3.1 for details on model learning task.
- **Select object for enrichment.** The user has the possibility to select an item for enrichment by browsing items in a given collection or by using the advanced search functionality. For object browsing, a collection available on the ASSETS server must be selected and the items will be retrieved by pressing the "Get Objects" button (see controls in red box). Alternatively, the user might search for items of a given data provider by providing a free text query, selecting the index fields to search for and pressing the "Search Objects" button (see controls in the blue box). The objects retrieved from the server will be displayed in the overview table, where basic information for the item identification is displayed (i.e. title, language, creator, year, uri). The item used for enrichment will be selected by clicking the corresponding row in the object table.
- **Perform enrichment & check results.** The enrichment is performed on the server after pressing the "Enrich Object" button and the result is displayed in the enrichment panel (see Figure 29). The knowledge extraction enrichment indicates the type of information inferred by the model, the text and the context in which this concept was detected (see the green box). The testing of the enrichment models should be performed by a user that is familiar with both the collections objects and the content of the training set. It is not guaranteed that an possible enrichment will be proposed for each collection object.



Europeana
Ingestion Control Panel

Test enrichment

This screen allows the testing of enrichment models!

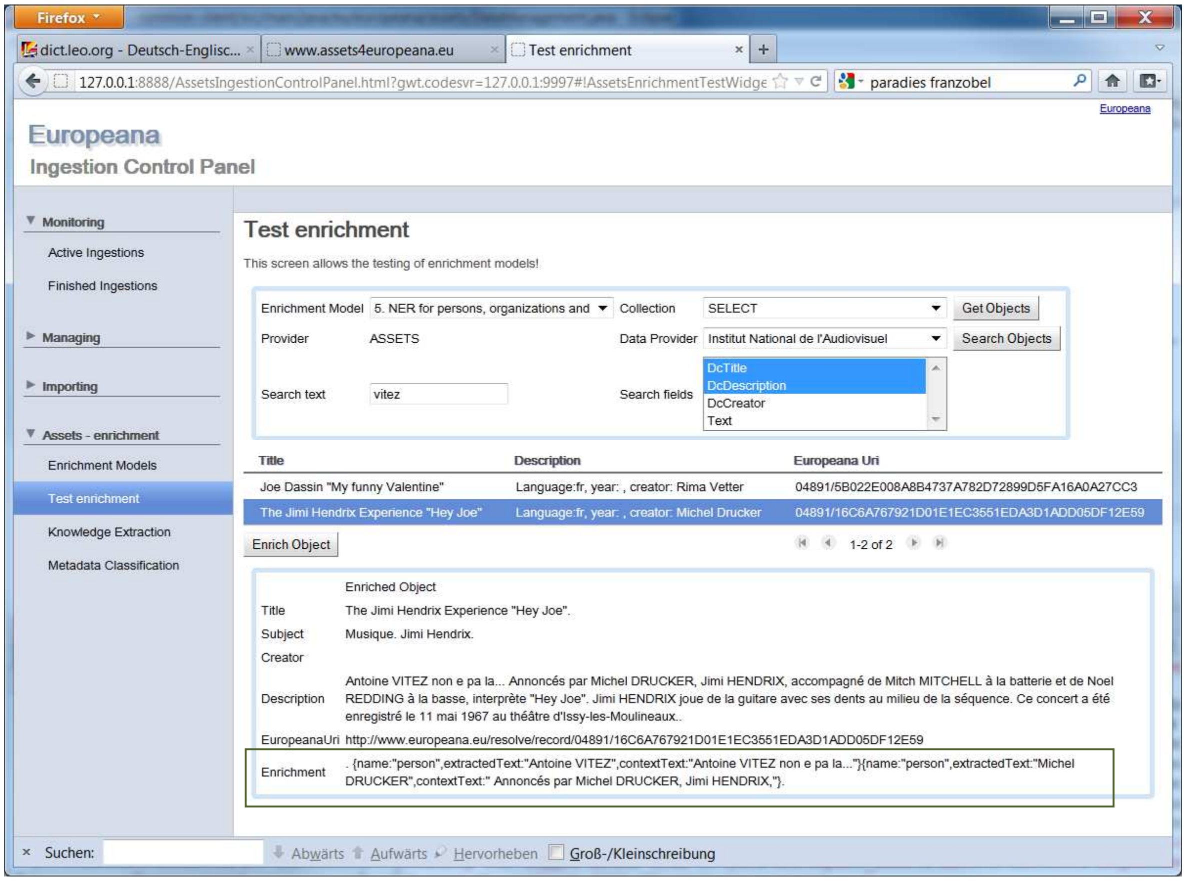
| | | | |
|---|---|---|---|
| Enrichment Model | 5. NER for persons, organizations and | Collection | SELECT |
| Provider | ASSETS | Data Provider | Institut National de l'Audiovisuel |
| Search text | vitez | Search fields | DcTitle, DcDescription, DcCreator, Text |

Get Objects
Search Objects

| Title | Description | Europeana Uri |
|---|---|---|
| Joe Dassin "My funny Valentine" | Language:fr, year: , creator: Rima Vetter | 04891/5B022E008A8B4737A782D72899D5FA16A0A27CC3 |
| The Jimi Hendrix Experience "Hey Joe" | Language:fr, year: , creator: Michel Drucker | 04891/16C6A767921D01E1EC3551EDA3D1ADD05DF12E59 |

Enrich Object
1-2 of 2

*Figure 28  Test enrichment screen – object selection*



Europeana
Ingestion Control Panel

Test enrichment

This screen allows the testing of enrichment models!

| | | | |
|---|---|---|---|
| Enrichment Model | 5. NER for persons, organizations and | Collection | SELECT |
| Provider | ASSETS | Data Provider | Institut National de l'Audiovisuel |
| Search text | vitez | Search fields | DcTitle, DcDescription, DcCreator, Text |

Get Objects
Search Objects

| Title | Description | Europeana Uri |
|---|---|---|
| Joe Dassin "My funny Valentine" | Language:fr, year: , creator: Rima Vetter | 04891/5B022E008A8B4737A782D72899D5FA16A0A27CC3 |
| The Jimi Hendrix Experience "Hey Joe" | Language:fr, year: , creator: Michel Drucker | 04891/16C6A767921D01E1EC3551EDA3D1ADD05DF12E59 |

Enrich Object
1-2 of 2

| | Enriched Object |
|---|---|
| Title | The Jimi Hendrix Experience "Hey Joe". |
| Subject | Musique. Jimi Hendrix. |
| Creator | |
| Description | Antoine VITEZ non e pa la... Annoncés par Michel DRUCKER, Jimi HENDRIX, accompagné de Mitch MITCHELL à la batterie et de Noel REDDING à la basse, interprète "Hey Joe". Jimi HENDRIX joue de la guitare avec ses dents au milieu de la séquence. Ce concert a été enregistré le 11 mai 1967 au théâtre d'Issy-les-Moulineaux.. |
| EuropeanaUri | http://www.europeana.eu/resolve/record/04891/16C6A767921D01E1EC3551EDA3D1ADD05DF12E59 |
| Enrichment | . {name:"person",extractedText:"Antoine VITEZ",contextText:"Antoine VITEZ non e pa la..."}{name:"person",extractedText:"Michel DRUCKER",contextText:" Annoncés par Michel DRUCKER, Jimi HENDRIX,"}. |

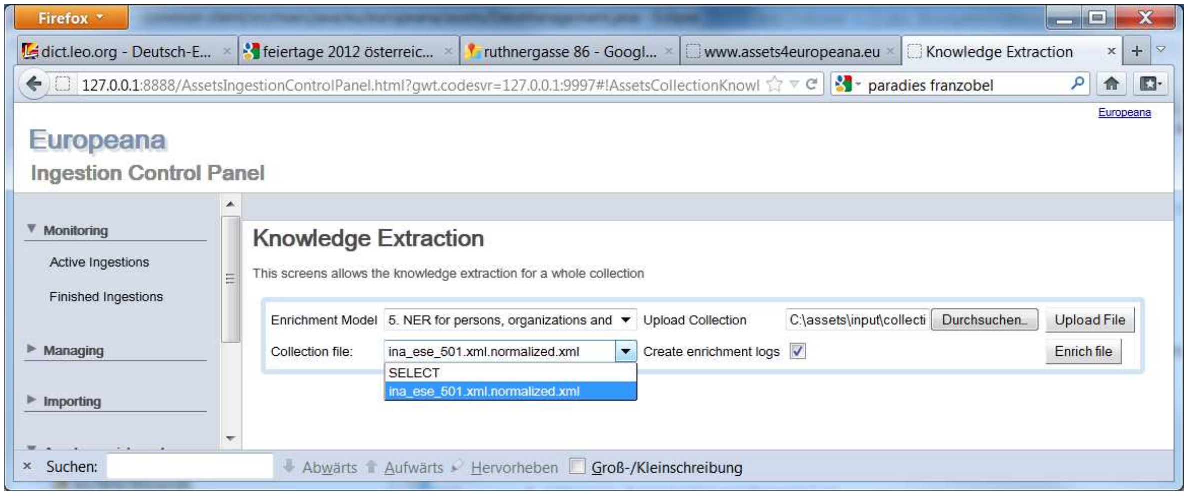
*Figure 29 Test enrichment screen – enrichment result*

### *5.3.3 Knowledge extraction screen*

The goal of the knowledge extraction service is to enhance the description of the collection objects by identifying concepts in the free text descriptions that can be linked to external resources like Thesauri or Wiki pages. This is quite a time consuming task given the amount of metadata available and the number of concepts available in the free text descriptions. Given the fact that the objects available in the collection are semantically related to each other, it is very alike that an enrichment model will be appropriate for processing all objects in a collection.

The knowledge extraction process consists of the execution of the following steps:

- **Upload metadata collection**. The whole knowledge extraction processing is performed on the ASSETS server; therefore, uploading the collection files to the server is a prerequisite for the execution of the enrichment process. This is accomplished in the GUI by providing a collection file and pressing the "Upload File" button (see Figure 30). The system does not allow uploading a metadata file two times on the server, but it allows performing the enrichment on the same collection files anytime it is needed.



Knowledge Extraction

This screens allows the knowledge extraction for a whole collection

Enrichment Model: 5. NER for persons, organizations and
Upload Collection: C:\assets\input\collecti
Collection file: ina_ese_501.xml.normalized.xml
Create enrichment logs: ✓
SELECT
ina_ese_501.xml.normalized.xml

*Figure 30 Knowledge extraction screen – enrichment invocation*

- **Knowledge extraction**. The metadata files available on the server are displayed in the Collection file combo box and can be used to perform the enrichment by selecting a suited enrichment model (only knowledge extraction models are displayed in the combo box) and pressing the "Enrich File" button. The user will be notified that the enrichment process has started and it may take several minutes until it will be finished. A second notification will appear when the enrichment is completed and a panel is displayed with the URLs available for downloading the enriched files (see Figure 31)

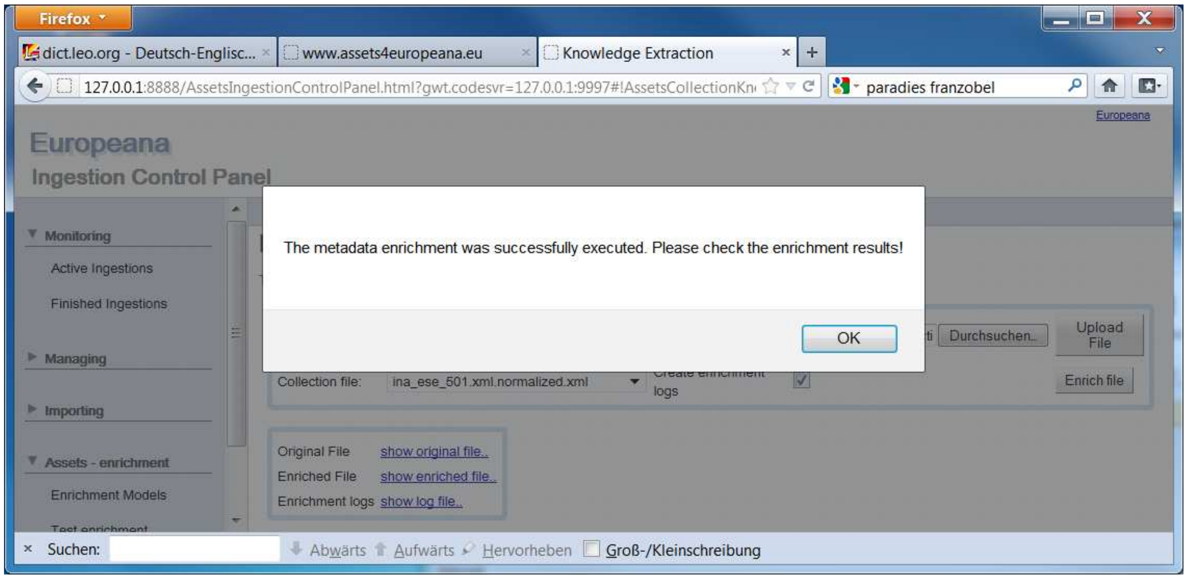
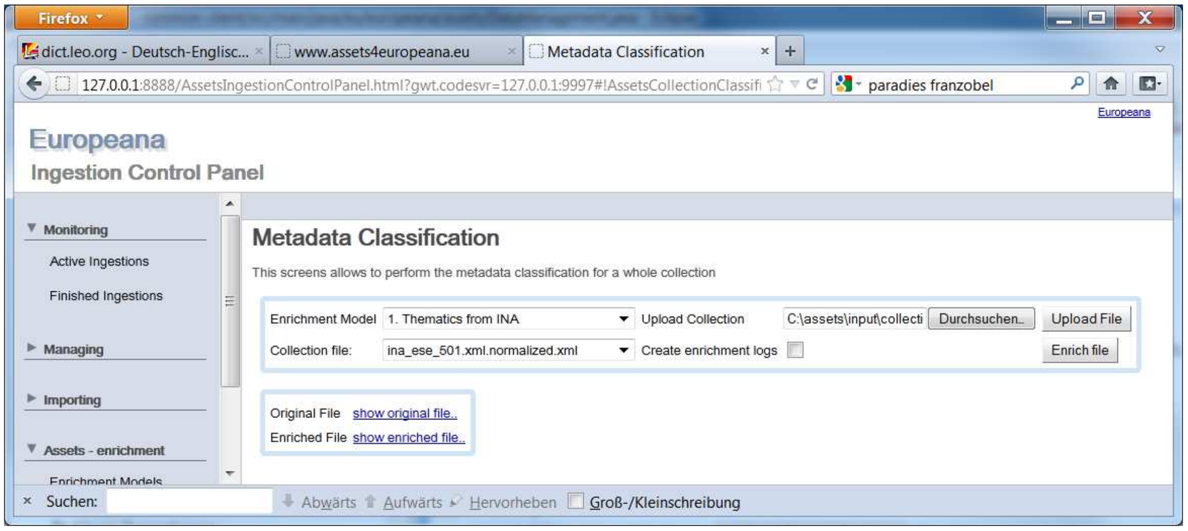

The metadata enrichment was successfully executed. Please check the enrichment results!

*Figure 31 Knowledge extraction screen – enrichment results*

### 5.3.4 Metadata classification screen

The metadata classification process is very similar to the knowledge extraction one (see 5.3.3). The same steps needs to be performed, but a different type of enrichment is applied (i.e. only classification models are available in this panel) and the result of the enrichment process will be available in a different ESE field (See also results presented in Section 5.3.2 ). Similar to knowledge extraction, the whole processing is performed on the server and the results are available for download as an xml file. The user needs to upload the metadata file (by using the "Upload File" button) on the server prior to performing the classification (to be launched by using the "Enrich File" button). The enrichment process can take a few minutes to be completed, but this process is typically performed faster that the knowledge extraction. The metadata classification screenshot is presented in Figure 32.


Metadata Classification

This screens allows to perform the metadata classification for a whole collection

Enrichment Model: 1. Thematics from INA
Upload Collection: C:\assets\input\collecti
Collection file: ina_ese_501.xml.normalized.xml
Create enrichment logs

*Figure 32  Metadata classification screen*

# 6. Evaluation of the services

Regarding the accuracy of the automatic enrichment functionalities provided by the services, an evaluation has been conducted and this section reports the evaluation results.

The evaluation uses datasets provided by the content providers of the ASSETS project and also from other Europeana content providers. Every dataset is composed by metadata records in which the information that is expected to be generated by the enrichment process is actually added to each metadata record by a human expert. The human-assigned values are used in the experiments for two purposes, as training data, or as test data, i.e., data with human assigned annotation are kept hidden to the automatic process and then compared to its outcome to determine the accuracy of the automatic process.

Whether a metadata record is used as training data or as test data depends on the adopted experimental protocols. The evaluation is based on two of the most common experimental protocols used in scientific literature. The choice of which protocol to use in a specific case depends on the dataset size.

For relatively small datasets (e.g., less than 1,000 metadata records), the adopted experimental protocol is *leave-one-out validation*. For a dataset of *n* metadata records, this protocol consists of running *n* training experiments using *n-1* metadata records as the training set, and then to apply the learned classification/extraction model to the held-out record. The classification/extraction responses for each metadata records are collected and then compared with the original ones in order to measure the accuracy.

For larger datasets, the adopted experimental protocol is *k-fold cross validation*. This protocol consists of splitting the dataset in k equally sized parts, and running *k* training experiments in which *k-1* parts of the data set are used as training set and the held out part as test set. The classification/extraction responses for each part are collected and then compared with the original ones in order to measure the accuracy.

As evaluation measure, we have adopted the $F_1$ measure for both the classification and extraction tasks. The $F_1$ measure (introduced in Section 2) is the standard evaluation measure in the domain of automatic text categorization and information extraction. We report here two usual versions of the measure: *micro*-averaged ($F_1^\mu$) and *macro*-averaged ($F_1^M$) already described in Section 2.

## 6.1 Knowledge extraction service

For the knowledge extraction service we have run experiments on the following datasets:

- INA: a collection of 10000 metadata records describing television recordings. The metadata records language is French, and the annotated concepts are Person, Organisation, and Location. Such a large number of records was available because the content provider had been already annotated this concepts in the metadata records before the ASSETS project.
- Albeniz: a collection of 75 metadata records describing musical recordings. The metadata records language is Spanish, and the annotated concepts are Person, Organisation, Location, Musical composition, and Award.

- DW: a collection of 268 metadata records describing audio and video recording of event reports/documentaries. The metadata records language is English, and the annotated concepts are Person, Organisation, and Location.

We have received samples of metadata records from other ASSETS partners, but any of those samples contained less than 30 records. With such a small number of records is  not suitable to perform a statistically relevant evaluation.

Table 1 shows the summary of the results obtained from the various experiments. We have selected various subsets within INA dataset to show how $F_1$ varies with respect to the training set size. The Albeniz collection obtained similar $F_1$ values with respect to the INA dataset of similar size. The DW collection, though with a smaller training set size than INA, obtains very good results.

After a manual inspection of the data, we have found that INA dataset in many cases suffers of partially annotation problem (e.g., only the last name of a person is annotated), while the DW dataset is almost free from this kind of errors, and very accurate.

Considering the results obtained on the various training sets sizes on the INA dataset and the high accuracy obtained by the higher quality DW, we can recommend a minimum training set size of 200-500 records, depending on the expected quality of the annotation.

| **Provider** | **Dataset description** | **Concepts** | **Number of metadata records** | **Experimental protocol** | **$F_1^\mu$** | **$F_1^M$** |
|---|---|---|---|---|---|---|
| INA | Collection of television recordings | Person, Organisation, Location | 10000 | 10-fold cross validation | .735 | .638 |
| INA | Collection of television recordings | Person, Organisation, Location | 5000 | 10-fold cross validation | .706 | .602 |
| INA | Collection of television recordings | Person, Organisation, Location | 1000 | 10-fold cross validation | .680 | .560 |
| INA | Collection of television recordings | Person, Organisation, Location | 500 | 10-fold cross validation | .540 | .461 |
| INA | Collection of television recordings | Person, Organisation, Location | 100 | leave-one-out | .440 | .232 |
| INA | Collection of television recordings | Person, Organisation, Location | 10 | leave-one-out | .121 | .065 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Albeniz | Collection of musical recordings | Person, Organisation, Location, Musical composition, Awards | 75 | leave-one-out | .466 | .182 |
| DW | Collection of video and audio recordings | Person, Organisation, Location | 268 | leave-one-out | .738 | .690 |

*Table 1: Results of experiments with the knowledge extraction service*

## 6.2 Metadata classification service

For the metadata classification service we have run experiments on the following datasets:

- INA: a collection of 10000 metadata records describing television recordings. The metadata records language is French, and the classification taxonomy is a set of 48 thematic areas (e.g. “Humour”, “Musique”). Such a large number of records was available because the content provider had already been classified its metadata records before the ASSETS project.
- Liberis: a collection of 6104 metadata records in Greek. Records are classified using a taxonomy with 6 thematic classes.
- ANSC: a collection of 15559 metadata records in Italian, describing audio/musical recordings. The classification taxonomy is composed of 522 genre-related classes (e.g., “Danze”, “Canto narrativo”, “Rito”, “Ninna nanna”).
- Albeniz: a collection of 75 metadata records describing musical recordings. The metadata records language is Spanish, and they are classified under 7 thematic classes.
- HASC: a collection of 1665 metadata records in Greek. Records describe artefacts, which are classified using a taxonomy with 32 classes (e.g., “Οχήματα - Επιβατικά αυτοκίνητα”: “vehicles – automobiles”).
- FLM: a collection of 786 metadata records in Italian. Records describe industrial artefacts and are classified using a taxonomy with 4 classes.
- CVCE: a collection of 17463 metadata records describing various media content (audio, text, image, video), with title and description expressed in multiple languages (English, French, German). They are linked to 1328 entries of the Eurovoc thesaurus, which is used as the classification schema.

Table 2 shows the summary of the results obtained from the various experiments.

Also for the classification service the experiments with different-sized training sets on the INA collection show that 500-1000 metadata records is a good lower limit for the training set size. Experiments on the other datasets also indicate that, when sizing the training set, the user has to take into account the number of classes that compose the taxonomy: the larger the taxonomy the larger should be the training set.

One relevant fact is that distribution of metadata records among classes usually follow a power law, with few classes with a large number of records assigned and many classes with just a few metadata records assigned. This effect is more evident when there are a large number of categories, with many of them represented by very few training examples.

The ANSC collection, for example, obtains a very low $F_1^M$ value, though it has many training examples, because about 200 classes out of 522 have less than 100 examples. On the contrary, the Albeniz collection, though very small, obtains a very high $F_1^M$ value because its 7 classes are almost uniformly distributed in the training set.

On the CVCE dataset we scored average quality results, which is however a positive results given the large number of classes involved.

| **Provider** | **Number of classes** | **Number of metadata records** | **Experimental protocol** | **$F_1^\mu$** | **$F_1^M$** |
|---|---|---|---|---|---|
| INA | 48 | 10000 | 10-fold cross validation | .647 | .300 |
| INA | 48 | 1000 | 10-fold cross validation | .568 | .240 |
| INA | 48 | 500 | 10-fold cross validation | .430 | .180 |
| INA | 48 | 100 | leave-one-out | .231 | .118 |
| Liberis | 6 | 6104 | 10-fold cross validation | .667 | .343 |
| ANSC | 522 | 15559 | 10-fold cross validation | .465 | .130 |
| Albeniz | 7 | 75 | leave-one-out | .842 | .839 |
| HASC | 32 | 1665 | 10-fold cross validation | .424 | .144 |
| FLM | 4 | 786 | leave-one-out | .993 | .745 |
| CVCE | 1378 | 17463 | 10-fold cross validation | .592 | .286 |

*Table 2: Results of experiments with the metadat classification service on collections provided by ASSETS partners.*

For the classification service, we have also been able to run evaluation experiments on some of the collections that are already part of Europeana, since some of them contained a consistent use of the *dc:subject* field to identify the topic of the content described by each metadata record using a controlled number of labels.

Results of this evaluation are reported in Table 3. In these cases all the collections have obtained good $F_1^{\mu}$ values but low $F_1^{M}$ values. After a manual inspection of the data, we motivate it by the fact that sometimes the dc:subject field contains spurious values related to other fields, e.g., the name of the author, and these values are not consistently assigned to records, resulting in training a classifier only for some classes with very poor training data.

| **Collection name** | **Taxonomy description** | **Number of classes** | **Number of metadata records** | **Experimental protocol** | $F_1^{\mu}$ | $F_1^{M}$ |
|---|---|---|---|---|---|---|
| DeutscheFotothek-1 | Photos/ Images | 1999 | 1097321 | 10-fold cross validation | .632 | .132 |
| DeutscheFotothek-2 | Photos/ Images | 220 | 529482 | 10-fold cross validation | .744 | .170 |
| landesarchiv | Photos/ Images | 2442 | 10407 | 10-fold cross validation | .292 | .191 |

*Table 3: Results of experiments with the metadata classification service on collections that are already part of Europeana.*

# 7. Concluding Remarks

The knowledge extraction and classification services, encapsulated into the ingestion workflow management, enable users to leverage on machine learning technique to automatically enrich the description of their collection objects.

The services described in this deliverable will support the enrichment processes to be used in application scenarios in which a user is willing to add well formatted structures to the information implicitly contained in free texts  Given a large collection, a user without access to the ingestion services has no alternative but to manually process the entire collection.  By using the ingestion services, the user can benefit the manual work done on a part of the collection to automatically process the rest of the collection.

We have run scientific experiments in order to determine the quality of the enrichment automatically performed by the services, finding that training set of about 500 metadata records can already produce quite an accurate output. Moreover, even in the case the output of the services is required to pass a stage of human inspection, the task of validating the automatic decisions taken by the services is much less demanding than performing manual annotation from scratch.

Finally, the outcome of the “evaluation of the ASSETS professional services” activity, performed by ASSETS content provider in order to enrich their metadata, will give further insight on the pros and cons of using automatic annotation tools in the ingestion workflow.

# 8. Appendix A

## 8.1 Traning set XSD schema for the knowledge extraction service

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://www.assets4europeana.eu/ExtractionSchema"
      targetNamespace="http://www.assets4europeana.eu/ExtractionSchema"
      elementFormDefault="qualified"
xmlns:ese="http://www.europeana.eu/schemas/ese/">
 <xs:import namespace="http://www.europeana.eu/schemas/ese/"
       schemaLocation="http://www.europeana.eu/schemas/ese/ESE-V3.3.xsd" />

 <xs:element name="extractionTrainingSet"
type="tns:extractionTrainingSetComplexType">
 </xs:element>

 <xs:complexType name="extractionTrainingSetComplexType">
  <xs:sequence>
   <xs:element name="extractionTask" type="tns:extractionTaskComplexType">
   </xs:element>
   <xs:element name="examples" type="tns:examplesComplexType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="extractionTaskComplexType">
  <xs:sequence>
   <xs:element name="sourceFieldName" type="xs:string"></xs:element>
   <xs:element name="conceptSet" type="tns:conceptSetComplexType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="conceptSetComplexType">
  <xs:sequence>
   <xs:element name="name" type="xs:string"></xs:element>
   <xs:element name="URI" type="xs:anyURI" minOccurs="0"></xs:element>
   <xs:element name="concepts" type="tns:conceptsComplexType"></xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="conceptsComplexType">
  <xs:sequence>
   <xs:element name="concept" type="tns:conceptComplexType"
         maxOccurs="unbounded">
```

```
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="conceptComplexType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"></xs:element>
    <xs:element name="URI" type="xs:anyURI" minOccurs="0"></xs:element>
    <xs:element name="targetField" type="xs:string" minOccurs="0"></xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="examplesComplexType">
  <xs:sequence>
    <xs:element name="example" type="tns:exampleComplexType"
             maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="exampleComplexType">
  <xs:sequence>
    <xs:element name="record" type="tns:recordComplexType">
    </xs:element>
    <xs:element name="extractedConcept" type="tns:extractedConceptComplexType"
             maxOccurs="unbounded"></xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="extractedConceptComplexType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"></xs:element>
    <xs:element name="extractedText" type="xs:string"></xs:element>
    <xs:element name="position" minOccurs="0">
      <xs:complexType>
        <xs:choice>
          <xs:element name="context" type="xs:string"></xs:element>
          <xs:sequence>
            <xs:element name="startCharacterPosition"
type="xs:positiveInteger"></xs:element>
            <xs:element name="endCharacterPosition"
type="xs:positiveInteger"></xs:element>
          </xs:sequence>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="URI" type="xs:anyURI" minOccurs="0"></xs:element>
  </xs:sequence>
  </xs:complexType>

  <xs:complexType name="recordComplexType">
    <xs:sequence>
      <xs:element ref="ese:record" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

## 8.2 Training set XML example for the knowledge extraction service

```
<?xml version="1.0" encoding="UTF-8"?>
<extractionTrainingSet xmlns="http://www.example.org/ExtractionSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/ExtractionSchema ExtractionSchema.xsd
">
  <!-- This section defines the extraction task performed on the records -->
  <extractionTask>
    <!-- This is the name of the field of the records from which information has to be
extracted -->
    <sourceFieldName>description</sourceFieldName>
    <conceptSet>
      <!-- This is a descriptive name for the extraction task -->
      <name>NER for persons, organizations and locations</name>
      <!-- Optional pointer to a resource that describes the concept set -->
      <URI>http://en.wikipedia.org/wiki/Named_entity_recognition</URI>
      <concepts>
        <concept>
          <!-- Name of the concept to be extracted -->
          <name>person</name>
          <!-- Optional pointer to a resource describing the concept -->
          <URI>http://en.wikipedia.org/wiki/Person</URI>
          <!-- Optional name of the target field in the record that has to be filled with
extracted information -->
          <targetField>extractedPerson</targetField>
        </concept>
        <concept>
          <name>organization</name>
          <URI>http://en.wikipedia.org/wiki/Organization</URI>
          <targetField>extractedOrganization</targetField>
        </concept>
        <concept>
          <name>location</name>
          <URI>http://en.wikipedia.org/wiki/Place_(geography)</URI>
          <targetField>extractedLocation</targetField>
        </concept>
      </concepts>
  </conceptSet>
 </extractionTask>
 <examples>
  <example>
   <record>
    <europeanaRecord>
     <title>Lamp</title>
     <description>Thomas Edison invented the filament lamp in America at almost the
same time as Joseph Swan did in England. He produced this type of lamp in 1880. This
particular bulb comes from Pullar's Dye Works in Perth, one of the first buildings in
Australia to install Edison lights.</description>
     <source>Tyne and Wear Imagine</source>
     <provider>CultureGrid ;  Uk</provider>

<identifier>http://www.imagine.org.uk/details/index.php?id=TWCMS:B5141a</identifie
r>
     <subject> inventors and innovators;  people</subject>
     <type>Image</type>
    </europeanaRecord>
   </record>
   <extractedConcept>
    <name>person</name>
    <extractedText>Thomas Edison</extractedText>
    <!-- A position specification is required when multiple instances of the extracted text
appear in the field with different role. In this case no position is required.-->
    <!-- In the case a position is necessary, it can be expressed by copying the extracted
text with enough surrounding text in order to make it uniquely identifiable. See examples
in  following concept extractions.-->
    <URI>http://viaf.org/viaf/66552944/#Edison, Thomas A. (Thomas Alva), 1847-
1931</URI>
   </extractedConcept>
   <extractedConcept>
    <name>location</name>
    <extractedText>America</extractedText>
    <!-- Also for this case the position is not required. It is just reported as an example. --
>
    <position>
     <context>lamp in America at almost</context>
    </position>
    <URI>http://www.geonames.org/maps/google_39.76_-98.5.html</URI>
   </extractedConcept>
   <extractedConcept>
    <name>person</name>
    <extractedText>Joseph Swan</extractedText>
    <!-- Position can also be expressed as the offset in number of characters from the
beginning of the text in the field. -->
    <position>
     <startCharacterPosition>80</startCharacterPosition>
```

```
      <endCharacterPosition>91</endCharacterPosition>
     </position>
     <URI>http://viaf.org/viaf/15100261/#Swan, Joseph Wilson, 1828-1914</URI>
    </extractedConcept>
    <extractedConcept>
     <name>location</name>
     <extractedText>England</extractedText>
     <URI>http://www.geonames.org/2635167/united-kingdom-of-great-britain-and-
northern-ireland.html</URI>
    </extractedConcept>
    <extractedConcept>
     <name>organization</name>
     <extractedText>Pullar's Dye Works</extractedText>

<URI>http://canmore.rcahms.gov.uk/en/site/127331/details/perth+pullar+s+dyeworks/
</URI>
    </extractedConcept>
    <extractedConcept>
     <name>location</name>
     <extractedText>Perth</extractedText>
     <URI>http://www.geonames.org/2063523/perth.html</URI>
    </extractedConcept>
    <extractedConcept>
     <name>location</name>
     <extractedText>Australia</extractedText>
     <URI>http://www.geonames.org/2077456/commonwealth-of-australia.html</URI>
    </extractedConcept>
    <extractedConcept>
     <name>person</name>
     <extractedText>Edison</extractedText>
     <position>
      <context>to install Edison lights</context>
     </position>
     <URI>http://viaf.org/viaf/66552944/#Edison, Thomas A. (Thomas Alva), 1847-
1931</URI>
    </extractedConcept>
   </example>
  </examples>
</extractionTrainingSet>
```

# 9. Appendix B

## 9.1 Traning set XSD schema for the metadata classification service

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://www.assets4europeana.eu/ClassificationSchema"
      targetNamespace="http://www.assets4europeana.eu/ClassificationSchema"
      elementFormDefault="qualified"
xmlns:ese="http://www.europeana.eu/schemas/ese/">
 <xs:import namespace="http://www.europeana.eu/schemas/ese/"
       schemaLocation="http://www.europeana.eu/schemas/ese/ESE-V3.3.xsd" />
 <xs:element name="classificationTrainingSet"
type="tns:classificationTrainingSetComplexType">
 </xs:element>

 <xs:complexType name="classificationTrainingSetComplexType">
  <xs:sequence>
   <xs:element name="classificationSchema"
type="tns:classificationSchemaComplexType">
   </xs:element>
   <xs:element name="examples" type="tns:examplesComplexType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="classificationSchemaComplexType">
  <xs:sequence>
   <xs:element name="name" type="xs:string"></xs:element>
   <xs:element minOccurs="0" name="URI" type="xs:anyURI">
   </xs:element>
   <xs:element name="type" default="multiLabel">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="multiLabel" />
      <xs:enumeration value="singleLabel" />
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="classes" type="tns:classesComplexType">
   </xs:element>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="examplesComplexType">
  <xs:sequence>
```

```
    <xs:element name="example" type="tns:exampleComplexType"
          maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="classesComplexType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="class"
          type="tns:classComplexType">
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="classComplexType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"></xs:element>
    <xs:element minOccurs="0" name="URI" type="xs:anyURI">
    </xs:element>
    <xs:element minOccurs="0" name="parentClassName" type="xs:string">
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="exampleComplexType">
  <xs:sequence>
    <xs:element name="record" type="tns:recordComplexType">
    </xs:element>
    <xs:element name="assignedClass" type="xs:string"
          minOccurs="0" maxOccurs="unbounded"></xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="recordComplexType">
  <xs:sequence>
    <xs:element ref="ese:record" />
  </xs:sequence>
</xs:complexType>

</xs:schema>
```

## 9.2 Training set XML example for the metadata classification service

```
<?xml version="1.0" encoding="UTF-8"?>
<classificationTrainingSet xmlns="http://www.example.org/ClassificationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/ClassificationSchema
ClassificationSchema.xsd ">
```

```
 <!-- This section defines the  -->
 <classificationSchema>
  <!-- A meaningful name for the classification schema -->
  <name>a simple music genre classificiation schema</name>
  <!-- Optional URI pointing to a description of the classification schema -->
  <URI>http://en.wikipedia.org/wiki/Music_genre</URI>
  <!-- Type of classification schema
     multiLabel: each record can be assigned to zero, one, or more than one class
     singleLabel: each record has to be assigned to one and only one class
  -->
  <type>singleLabel</type>
  <!-- List of classes -->
  <classes>
   <!-- Definition of a class -->
   <class>
    <!-- Class name -->
    <name>classical</name>
    <!-- Optional URI describing the class -->
    <URI>http://en.wikipedia.org/wiki/Classical_music</URI>
   </class>
   <class>
    <name>baroque</name>
    <URI>http://en.wikipedia.org/wiki/Baroque_music</URI>
    <!-- The classification schema could be hierachical. In this case "baroque" is a more
specific definition of a classical music genre. -->
    <!-- If a record is assigned to baroque it is implicitly also an example of classical
music. -->
    <!-- If a record is assigned to classical it means that, though the record refers to
classical music,  it does not belong to anyone of the more specific classes. -->
    <parentClassName>classical</parentClassName>
   </class>
   <class>
    <name>modern</name>
    <URI>http://en.wikipedia.org/wiki/20th-century_classical_music</URI>
    <parentClassName>classical</parentClassName>
   </class>
   <class>
    <name>romantic</name>
    <URI>http://en.wikipedia.org/wiki/Romantic_music</URI>
    <parentClassName>classical</parentClassName>
   </class>
   <class>
    <name>jazz</name>
    <URI>http://en.wikipedia.org/wiki/Jazz</URI>
   </class>
   <class>
    <name>bebop</name>
    <URI>http://en.wikipedia.org/wiki/Bebop</URI>
```

```
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>funky</name>
      <URI>http://en.wikipedia.org/wiki/Funk</URI>
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>swing</name>
      <URI>http://en.wikipedia.org/wiki/Swing_music</URI>
      <parentClassName>jazz</parentClassName>
    </class>
    <class>
      <name>popular</name>
      <URI>http://en.wikipedia.org/wiki/Popular_music</URI>
    </class>
    <class>
      <name>country</name>
      <URI>http://en.wikipedia.org/wiki/Country_music</URI>
      <parentClassName>popular</parentClassName>
    </class>
    <class>
      <name>punk</name>
      <URI>http://en.wikipedia.org/wiki/Punk_rock</URI>
      <parentClassName>popular</parentClassName>
    </class>
    <class>
      <name>rap</name>
      <URI>http://en.wikipedia.org/wiki/Hip_hop_music</URI>
      <parentClassName>popular</parentClassName>
    </class>
  </classes>
</classificationSchema>
<examples>
  <example>
    <record>
      <recording>
        <title>The Marriage of Figaro</title>
        <author>Wolfgang Amadeus Mozart</author>
        <year>1943</year>
        <director>Paul Breisach</director>
        <orchestra>Metropolitan Opera Orchestra</orchestra>
        <location>New York</location>
      </recording>
    </record>
    <assignedClass>classical</assignedClass>
  </example>
  <example>
```

```
      <record>
        <recording>
          <title> Brandenburg Concerto No. 1 - 1</title>
          <author>Johann Sebastian Bach</author>
          <orchestra>The Busch Chamber Players</orchestra>
          <URI>http://en.wikipedia.org/wiki/File:Bach_-_Brandenburg_Concerto_No._1_-
_1._Allegro.ogg</URI>
        </recording>
      </record>
      <assignedClass>baroque</assignedClass>
    </example>
    <example>
      <record>
        <recording>
          <title>Für Elise</title>
          <title>Bagatelle No. 25 in A minor</title>
          <author>Ludwig van Beethoven</author>
          <year>1867</year>
        </recording>
      </record>
      <assignedClass>romantic</assignedClass>
    </example>
    <example>
      <record>
        <recording>
          <title>Salt Peanuts</title>
          <composed>1942</composed>
          <year>1947</year>
          <author>Dizzy Gillespie</author>
          <description>Dizzy played for Lucky Millinder's band in the early '40s. It was a riff
this band played, after a Dizzy solo in the tune "Little John Special", that Dizzy developed
into his tune "Salt Peanuts".</description>
          <URI>http://www.youtube.com/watch?v=kOmA8LOw258</URI>
        </recording>
      </record>
      <assignedClass>bebop</assignedClass>
    </example>
    <example>
      <record>
        <recording>
          <title>Gotta Lotta Love</title>
          <author>Tracy Marrow</author>
          <author>Ice-T</author>
          <album>Home invasion</album>
          <year>1993</year>
        </recording>
      </record>
      <assignedClass>rap</assignedClass>
    </example>
    <example>
      <record>
        <recording>
          <title>The river unbroken</title>
          <author>Dolly Rebecca Parton</author>
          <year>1987</year>
          <album>Rainbox</album>
          <label>CBS</label>
          <producer>Steve "Gold-E" Goldstein</producer>
        </recording>
      </record>
      <assignedClass>country</assignedClass>
    </example>
  </examples>
</classificationTrainingSet>
```