Mastering Data-Intensive Collaboration and Decision Making

FP7 - Information and Communication Technologies
Grant Agreement no: 257184
Collaborative Project
Project start: 1 September 2010, Duration: 36 months

D3.2.1 – The Dicode Data Mining Services (initial version)

Due date of deliverable:	31 August 2011
Actual submission date:	31 August 2011
Responsible Partner:	NEO, FHG
Contributing Partners:	NEO, FHG

Nature: ☒ Report ☒ Prototype ☐ Demonstrator ☐ Other

Dissemination Level:

☒ PU : Public
☐ PP : Restricted to other programme participants (including the Commission Services)
☐ RE : Restricted to a group specified by the consortium (including the Commission Services)
☐ CO : Confidential, only for members of the consortium (including the Commission Services)

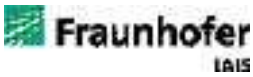
Keyword List: Data Mining Services, Abstract Description.

The Dicode project (dicode-project.eu) is funded by the European Commission, Information Society and Media Directorate General, under the FP7 Cooperation programme (ICT/SO 4.3: Intelligent Information Management).

The Dicode Consortium

Research Academic Computer Technology Institute (CTI) (coordinator), Greece

University of Leeds (UOL), UK



Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e.V. (FHG), Germany

Universidad Politécnica de Madrid (UPM), Spain

Neofonie GmbH (NEO), Germany

Image Analysis Limited (IMA), UK

Biomedical Research Foundation, Academy of Athens (BRF), Greece

Publicis Frankfurt Zweigniederlassung der PWW GmbH (PUB), Germany

Document history			
Version	Date	Status	Modifications made by
1	30-07-2011	First Draft	Jörg Kindermann, FHG Doris Maassen, NEO Axel Poigné, FHG
2	15-08-2011	Second draft (version sent to internal reviewers)	Ahmad Ammari, UOL Georgia Tsiliki, BRF
3	25-08-2011	Third draft (evaluation comments incorporated, version sent to SC)	Doris Maassen, NEO Axel Poigné, FHG
4	29-08-2011	SC comments incorporated	Nikos Karacapilidis, CTI Lydia Lau, UOL Guillermo de la Calle Velasco, UPM
5	31-08-2011	Final (approved by SC, sent to the Project Officer)	Axel Poigné, FHG

Deliverable manager

- Axel Poigné, FHG

List of Contributors

- Jörg Kindermann, FHG
- Doris Maassen, NEO

List of Evaluators

- Ahmad Ammari, UOL
- Georgia Tsiliki, BRF

Summary

The deliverable has dual nature, it reports on the status of development of the Dicode Data Mining Services and it consists of the actual prototypes of these services. The services are presented in form of abstract descriptions, i.e. high-level descriptions of the functionalities provided by the services. The implementation status of the services is indicated and example outputs are provided as appendix. The abstract descriptions are complemented by tentative rules for implementation as RESTful services aiming for uniformity of the application interfaces.

Table of Contents

1 Introduction

The Dicode Data Mining Services comprise a set of services addressing the data mining challenges within the Dicode project as presented in the "Data Mining Framework" (Deliverable 3.1.1). Subsequently, a coarse description of Dicode Services is provided in textual format according to the template below.

Name	Name of the Dicode Service or Interface Type *Convention*: All individual words in the service type name are capitalised.
Standards	Reference to an abstract or a platform-specific service specification according to a standardisation organisation (e.g. ISO, CEN, W3C, OGC, ...) or to reference material that has been taken into account when describing the service, its interfaces or operations.
Description	Description of the functionality provided by the Dicode service or interface. The service description shall end with: The <name> Service provides its functionality through the following interfaces: $Interface_1$: Description of the purpose of interface 1 … $Interface_N$: Description of the purpose of interface N *Note:* If an interface is re-used from another Dicode Service Type description, the name of this service type shall be indicated in brackets in the interface definition below.
Interface	$Interface_1$
Opera	Description of the operation i of the interface. Only major input and output information shall be described, no individual request and result parameters. Optional operations are to be marked by suffix (optional) after the operation name.
…	
Interface	$Interface_N$
…	
Example usage	Description of an example usage scenario of the service.
Comments	Description of current restrictions or possible extensions and enhancements in future versions.
Conformance classes	if defined
Implementation rules	Reference to the corresponding implementation rules if defined.
Implementation status	Information about the current status of the implementation of the service, eventually reference to examples of outputs in Appendix B.
UML model	References to related UML models if available.

The presentation demonstrates the basic functionality of a service in terms of interfaces and operations. These "abstract service descriptions" are meant as a high-level introduction to Dicode Data Mining Services and will be complemented by formal abstract (i.e. platform-neutral) specification (e.g., in form of UML diagrams) and/or (machine-readable) implementation specifications when the services become fully operational.

The abstract service descriptions indicate the implementation status. A number of services are in prototype status while others are still in design status. For the prototypes, examples of output data are provided. The output and input formats are still in development. Hence, outputs are provided in a format accessible for a human reader. Prototypes of the services still in design status will be available in M24. All services will be operational by end of the project. Experience gained by experiments may trigger modifications and additional requirements not yet foreseen.

The abstract service descriptions are complemented by descriptions of the general principles for service mapping, i.e. the binding of a service's interfaces and operations to functionality as provided by a specific "service platform". A "service platform" comprises all the necessary elements to run a service (e.g. W3C Web services, RESTful services). At present, Dicode targets RESTful services.

2 Abstract Service Descriptions

2.1 Text Mining Services

2.1.1 Twitter Harvester

In Deliverable D3.1.1 "The Dicode Data Mining Framework", the Twitter Harvester was introduced as a data acquisition component of the Dicode Framework which focuses exclusively on Twitter data. The Twitter Harvester saves data obtained from Twitter for research purposes.

Twitter offers two different ways to obtain tweets (status updates): Twitter's Streaming API[1] and its Search API[2]. The Twitter Harvester uses both APIs to build up a corpus of twitter data that can be used by the Dicode Data Mining Services according to Twitter's term of service[3]. The Twitter Harvester will as well conform to Twitters' "Rules of the Road for Developers"[4] that pose severe restrictions on the types of services that can be implemented based on the API:

- "Your Client must use the Twitter API as the sole source for features that are substantially similar to functionality offered by Twitter. Some examples include trending topics, who to follow, and suggested user lists."
- "You may not use Twitter Content or other data collected from end users of your Client to create or maintain a separate status update or social network database or service." [5]

This implies that Dicode is not entitled to provide a search service for tweets as it is offered by the Twitter search API[6]. For direct access of Twitter data, for example for a search for tweets, Dicode will use Twitter's Search API[7] directly. The Twitter Harvester does not have to be considered as a "separate status update database" according to the "Rules of the Road for Developers" because it is only used as an internal resource for the generation of derived data by other services of the Dicode Data Mining Framework.

The Twitter Harvester can be configured via a RESTful API. Service users will be able to configure a set of queries which will be fired to Twitter's Search API regularly without exceeding the rate limit.[8]

Name	Twitter Harvester
Standards	Twitter provides no XML schema for the Twitter search API or streaming API. For secure API authorisation Twitter uses OAuth.[9] [10] The Twitter Harvester uses the widely popular 3rd party library Twitter4J[11] for Twitter

[1] http://dev.twitter.com/pages/streaming_api
[2] https://dev.twitter.com/docs/api
[3] http://twitter.com/tos.
[4] https://dev.twitter.com/terms/api-terms
[5] https://dev.twitter.com/terms/api-terms
[6] Twitters' search interface uses the main parameters of the search API: http://twitter.com/#!/search-advanced.
[7] http://dev.twitter.com/doc/get/search
[8] http://dev.twitter.com/pages/rate-limiting

	access.
Description	The Dicode Twitter Harvester Service is a data acquisition component which collects twitter status updates. The service allows for configuration of queries to Twitter's search API which are regularly executed for Twitter harvesting. This service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *Configuration*: Interface for the configuration of Twitter Harvester at runtime
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the configured queries.
Interface	*Configuration*
listQueries	Shows all configured queries
addQuery	Adds a query to the configuration
editQuery	Edit a configured query
removeQuery	Removes a query from the configuration
Example usage	Add a query to the configured set of queries, e.g. the term "Dicode" to acquire all statuses which deal with the Dicode project.
Comments	Twitter Harvester acquires tweets using two different mechanisms: In some use cases we need a representative sample[12] of all public status updates which is provided by Twitters Streaming API. But if we are looking for tweets about "Dicode", the sample stream will not return enough results and we have to query the Twitter Search API. For the Data Mining Services proposed in D3.2.1 the 1% sample will suffice. The implementation of the configuration component of the Twitter Harvester will therefore be postponed.[13] Both harvesting mechanism internally use the same data storage interface which is based on Twitter4J. The current prototype of Twitter Harvester serves as a data provider for the Dicode Twitter Pre-processing Service.
Conformance classes	The service has to conform to Twitter's rate limit.
Implementation rules	Not yet available.
Implementation status	Prototypical version implemented. For a snapshot of output data, see Appendix B.1
UML model	Not yet available.

[9] http://oauth.net/
[10] http://tools.ietf.org/html/rfc5849
[11] http://twitter4j.org
[12] http://dev.twitter.com/pages/streaming_api_concepts#sampling

2.1.2 Data pre-processing

As described in D.3.1.1 "The Dicode Data Mining Framework", data pre-processing features several operations which will be part of more complex components. D3.1.1 distinguishes between more general operations which will be required for several data sources and more specific operations which depend on the data source.

The first pre-processing service implemented provides rather generic operation on a specific data source, namely Twitter data.

Pre-processing of Twitter data

The Twitter Pre-processing Service returns a bag-of-words representation of Twitter status updates (tweets). Processing and export of Twitter data is triggered via the same service interface.

The service employs Yahoo's Hadoop[14] Workflow Engine "Oozie"[15] for MapReduce job chaining. The workflow, which is easily extensible, currently consists of two steps:

- Export of Twitter data acquired by the Twitter Harvester which is stored in a HBase[16] column database in the Hadoop Distributed File System (HDFS)[17]
- Transformation of the exported data into a term-frequency vectors.

Both steps can be configured via service parameters (see description below). The pre-processing of Twitter data is performed in batch mode. The final results of the pre-processing process will be made available in a compressed binary format which can be consumed by the Dicode Topic Service. The data pre-processing operations of Twitter data are only available for internal use, because Dicode is not entitled to redistribute Twitters status updates.

Name	Twitter Pre-processing
Standards	Hadoop Sequence:[18] http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/SequenceFile.html
Description	The Dicode Twitter Pre-processing Service exports Twitter status updates (tweets) stored in a HBase column store. The output of the service is a compressed archive which contains Hadoop sequence files of serialized tweets in Mahouts vector format and a dictionary file mapping numerical term identifiers to terms. This service provides its functionality through the following interfaces:

[13] We currently call the Twitter Search API for evaluation purposes via shell scripts scheduled by the Cron daemon.
[14] http://hadoop.apache.org/
[15] http://yahoo.github.com/oozie/
[16] http://hbase.apache.org/
[17] HBase is built on top of HDFS. Hadoop's distributed file system HDFS provides the replication for the data blocks which store the HBase data tables.

	• *ServiceCapabilities*: Informs about the common and specific capabilities. • *Asynchronous Interaction:* Supports the invocation as asynchronous operation. • *TwitterVectorExport*: Exports term-frequency vectors of a selected set of tweets.
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities.
Interface	*Asynchronous Interaction*
invokeAsync	Invoke operation
abort	Abort operation
Interface	*TwitterVectorExport*
getTweets	Exports a set of tweets as term-frequency Vectors. The selection of tweets from the column store can be configured by the following filters: query (a regular expression to match the tweet text), language, publication date or time span. The term-frequency vector output can be configured via the parameter "minSupport" which defines the minimal occurrence of a term in the selected set of tweets.
Example usage	The service provides tweets for further processing, e.g. for the Dicode Topic Detection Service
Comments	Currently the output of the service conforms to the required input format of the topic detection services. If needed different text processing algorithms like e.g. tf-idf weighting will be supplied.
Conformance classes	Not yet available.
Implementation rules	Not yet available.
Implementation status	Prototypical version implemented. For a snapshot of output data, see Appendix B.2.
UML model	Not yet available.

2.1.3 Automatic Tagging

The envisioned auto-tagging service which was described in D3.1.1 returns relevant catchphrases from text. The proposed algorithm uses different kinds of features for the elicitation of catchphrase candidates, e.g. the position of the phrase, the existence of the phrase in Wikipedia etc. For the analysis of Twitter status updates, this procedure is neither reasonable nor feasible: containing a maximum of 140 letters, tweets are too short – and Twitter users usually mark the most important keywords with the hash prefix[19], for example #dicode. For D3.2.1 we therefore offer a first version of a "Keytrends Service", which makes use of the available metadata. The service currently returns the top hashtags for a given topic. Addi-

[18] https://cwiki.apache.org/MAHOUT/creating-vectors-from-text.html
[19] http://support.twitter.com/articles/49309-what-are-hashtags-symbols

tional experiments with other types of Twitter meta-data like the language, country, user identifier, retweet count etc. are being conducted.[20]

In the first implementation, the service aggregates hashtags of Twitter status updates stored in the HBase column store. The service provides the top 100 hashtags for a given topic and time and their frequencies.

Name	Keytrends Service
Standards	JSON (JavaScript Object Notation)[21]
Description	The Dicode Keytrends Service delivers a variety of meta-data for the asseesment of key trends. The first version returns aggregated meta-data about Twitter status updates which describe key trends for a given topic, place and time. The Keytrends Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *TwitterMetadata*: Provides aggregated metadata about the microblogging sphere
Interface	ServiceCapabilities
getCapabilities	Informs the requestor about the available operations of the service.
Interface	*TwitterMetadata*
getHashtags	This operation returns the top hashtags from a subset of tweets. Currently a maximum of 100 top hashtags is returned. For tweet selection a set of filters can be applied, e.g. filter by topic, language and publication date.
listQueries	Shows all configured queries
addQuery	Adds a query to the configuration
editQuery	Edit an existing query
removeQuery	Removes a query from the configuration
Example usage	The service can be used for key trend detection in Social Media Monitoring (Use case 3).
Comments	The Twitter metadata returned from this service is generated in batch mode. If the analyst wants to use the results interactively – for example in a web application – he/she has to set up the data analysis jobs first. A typical use case will be brand monitoring where the analyst defines the data set via a set of filters and the analysis interval (e.g. daily). In the future the Keytrends Service will offer a variety of interfaces for key trend assessment. The interfaces have not been specified yet.
Conformance classes	Not yet available.

[20] Twitter offers a huge amount of meta-data – even the background colour of the user's Twitter profile is available. Only the most relevant meta-data will be aggregated for the Keytrends Service.

[21] http://www.json.org/

Implementation rules	Not yet available.
Implementation status	Prototypical version implemented. For a snapshot of output data, see Appendix B.3.
UML model	Not yet available.

2.1.4 Topic detection

This service is intended to give the user a quick albeit superficial overview of the thematic content of a document collection. Here, large document collections of twitter messages and/or collections of consumer discussions in Internet forums have to be scanned for the thematic content that is interesting for the analyst. Each of the detected topics is described by a list of keywords sorted by keyword importance (i.e. probability of occurrence) for this topic.

The service is based on Latent Dirichlet Allocation (LDA) [22]. One very attractive point of LDA is that it effectively generates low-dimensional representations from sparse high-dimensional data, representing documents by a low-dimensional vector of topics. LDA is a Bayesian probabilistic model that describes document corpora in a fully generative way[23]. It assumes a fixed number K of underlying topics in a document collection D, where each document d is a mixture of topics. According to LDA, the observable variables, i.e., the words in a document, are generated as follows: First, for each document d, document-specific topic proportions are drawn. Then for each word i in d a topic z_{di} is randomly chosen. Finally, the observed word w_{di} is randomly drawn from the distribution of the selected topic. This overall topic distribution is assumed to be drawn randomly beforehand from a Dirichlet distribution.

The resulting word distributions for each topic have high probabilities for words that often co-occur in documents. Topic models alleviate two main problems arising in natural language processing: synonymy and polysemy. Synonymy refers to the case where two different words (say car and automobile) have the same meaning. Synonyms usually will co-occur in similar contexts and hence belong to similar topics. Polysemy, on the other hand, refers to the case where a term, such as plant, has multiple meanings (industrial plant, biological plant). Depending on the context at hand (e.g., the document is mostly about industry or biology) different topics will be assigned to the word plant.

Name	Topic Detection
Standards	Term-frequency vectors (as input)
Description	The Dicode Topic Detection Service gives the user a quick albeit superfi-

[22] Blei, DM, Ng, AY, Jordan, MI, Latent Dirichlet Allocation, The Journal of Machine Learning Research, 2003 vol. 3 pp. 993-1022.
[23] Blei, DM, Ng, AY, Jordan, MI, Latent Dirichlet Allocation, The Journal of Machine Learning Research, 2003 vol. 3 pp. 993-1022.

	cial overview of the thematic content of a document collection. The Topic Detection Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *Detection:* Provides the topics
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*Detection*
provideTopics	Provides *n* topics as described by a list of *m* keywords per topic. Parameters: number of topics, number of keywords per topic, term-frequency vectors that have been computed from the original input texts, list of stop-words that will be removed from the term frequency vectors (optional) Output is a comma separated table of topics. Each topic corresponds to a row in the table. Each row consists of a list of keywords sorted by keyword importance.
Example usage	Topic detection will be applied in Use Case 3 to provide an overview of the thematic content of a text or tweet collection. A second application are is possibly topic detection in bio-medicinal texts with regard to Use Case 1 or 2
Comments	• The input of term frequency vectors is obtained from the Twitter Pre-processing service. • At present, stop-words are used optional input for practical reasons since they are at present difficult to use for the Twitter Pre-processing Service since not supported by the Mahout framework yet. • Weights (probabilities) may be added to the keywords. • Since the number of topics detected in examples is very high, a hierarchical approach may be considered: (1) Documents are chosen using pattern matching and language filters (2) N topics are computed (3) Choose one of the N topics (4) Compute M new topics on the basis of documents that relate to the topic chosen in (3) (5) eventually iterate with (3) with choosing a topic from the first N topics or of the second-level (nth-level) M topics
Conformance classes	Not yet available.
Implementation rules	Not yet available.

Implementation status	Prototypical version of the algorithm is implemented. Not yet provided as service. For a snapshot of output data see Appendix B.4
UML model	Not yet available.

2.2 Designated Text Mining Services

For the following text mining services, only tentative abstract descriptions are provided since implementation work has not yet started. These services are complementary, in that they all represent different facets of opinion mining.

2.2.1 Opinion Mining

Textual information in the world can be broadly categorized into two main types: facts and opinions. Facts are objective expressions about entities, events and their properties. Opinions are usually subjective expressions that describe people's sentiments, appraisals or feelings toward entities, events and their properties. The concept of opinion is very broad. The Opinion Mining Service will aggregate results obtained from named entity recognition, sentiment analysis, emotion detection, and relation extraction using machine learning techniques. Which concrete algorithms will be used needs evaluation but depends on availability of the services mentioned above.

The intention is to extract opinions that texts considered might have with respect to products, brands, or related topics. The development will be based on FHG technology.

Name	Opinion Mining
Standards	No relevant standards available
Description	The Dicode Opinion Mining Service The Opinion Mining Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *OpinionMining*: provides the opinion mining functionality
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*OpinionMining*
provideOpinions	Provides opinions about products, brands, or related topics
Example usage	Will be applied in use case 3 to give an overview of opinions expressed in a text collection
Comments	• Opinion mining will be applied to Twitter tweets as text corpora. • A format for output will be designed depending on the aggregation algorithms. • The service will be an orchestration of the Named Entity Recognition, Sentiment Analysis, Emotion Detection, and Relation Extraction Services
Conformance clas-	Not yet available.

ses	
Implementation rules	Not yet available.
Implementation status	Not yet implemented
UML model	Not yet available.

2.2.2 Named Entity Recognition

Named entity recognition (NER) is the task of the identification and classification of proper names in natural-language text. A named entity is defined as a unique real world entity denoted by a proper noun or name. Entities can be grouped in different classes. Among them are

- Person: a named person (e.g. a politician, an artist)
- Location: a politically or geographically defined location (e.g. cities, countries, etc.)
- Organisation: an organizational entity (e.g. United Nations, Volkswagen)

NER has to annotate the tokens of a text, i.e. words, punctuation marks, or other atomic parse elements as members of the named entity of interest. Tokens that are not part of a named entity are annotated as "Other". One challenge of this task is that a token's entity type need not be unique but depends on the entities context. For example, the entity "Germany" can be a location in a geographical context but also an organization in political context. In the biological domain, we have for instance the entity classes "DNA" or "Protein" and the same entity may also belong to different classes.

Named entities often consist of groups of tokens and in many cases the presence or absence of one token indicates different entities. For example, the term "Arsenal" denotes a part of London and has therefore the entity type location, whereas "FC Arsenal" is an organization. On the other hand, names can be abbreviated, either to acronyms that are typical for organizations (e.g. "Arsenal" for the football club) or surnames in case of persons (i.e. "Obama" instead of "Barack Obama").

Named entity recognition will use conditional random fields based on extracted syntactical structures.

Name	Named Entity Recognition
Standards	No relevant standards available
Description	The Dicode Named Entity Recognition Service identifies terms such as proper names, geographical locations, enterprise names, or commercial products in the texts and assigns a class label (like "name", "location", etc.). An alternative option is not only to annotate named entities in the classical sense but also to identify key phrases that are typical for the domain. The Named Entity Recognition Service provides its functionality through the following interfaces:

	• *ServiceCapabilities*: Informs about the common and specific capabilities. • *NER*: provides the named entity extraction functionality
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*NER*
annotate	Provides annotated texts. Parameters: a collection (corpus) of full texts and the name of annotation category system. Output consists of texts with named entities annotated.
Example usage	Will be applied in use case 3 to identify persons, locations, brand names, etc. Moreover, named entity could be adapted for annotation of genes and other biological entities in Use Case 1. In this case, the inputs and outputs may have a format different from texts.
Comments	• Sentiment analysis will be applied to Twitter tweets as text corpora. • The format of text annotation needs to be determined. • Output used for the Sentiment Analysis, Emotion Detection, and Relation Extraction Services
Conformance classes	Not yet available.
Implementation rules	Not yet available.
Implementation status	Not yet implemented
UML model	Not yet available.

2.2.3 Sentiment Analysis

The intention is to identify text passages that include sentiments, and – if possible – also indicates the nature (positive versus negative or further categories) of the sentiment and, at a later stage, polarity and weight of sentiments. Sentiments can be identified according to contextual regularities, much in the same way as named entities (of course, there are also sophisticated ways to express one's sentiments between the lines; the more complicated cases of sentiment expression are not yet accessible to automated analysis).

Sentiment analysis will use machine-learning algorithms for classification (typically based on support vector machine as well as specific algorithms developed by FHG).

Name	Sentiment Analysis
Standards	No relevant standards available
Description	The Dicode Sentiment Analysis Service identifies text passages that include sentiments, and – if possible – also indicates the nature (positive versus negative or further categories) of the sentiment.

	The Sentiment Analysis Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *SentimentAnalysis*: provides the named sentiment analysis functionality
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*SentimentAnalysis*
annotate	Provides annotated texts. Inputs are text corpora. Output is the same as input but with sentiments annotated.
Example usage	Will be applied in use case 3 to identify sentiments expressed in the texts
Comments	• Sentiment analysis will be applied to Twitter tweets as text corpora. • The format of text annotation needs to be determined. • The service may use ontologies such as sentiWordNet. • The Sentiment Analysis Service will eventually use the output of the Named Entity Recognition Service.
Conformance classes	Not yet available.
Implementation rules	Not yet available.
Implementation status	Not yet implemented
UML model	Not yet available.

2.2.4 Emotion Detection

This service detects some types of emotions such as satisfaction, anger, surprise, etc. This is still leading edge research, so the results are not fully predictable.

Sentiment analysis will use machine-learning algorithms for classification (typically based on support vector machine as well as specific algorithms developed by FHG).

Name	Emotion Detection
Standards	No relevant standards available
Description	The Dicode Emotion Detection Service The Emotion Detection Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capa-

	bilities. • *EmotionDetection*: provides the emotion detection functionality
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*EmotionDetection*
annotate	Provides annotated texts Inputs are text corpora. Output is the same as input but with emotions annotated
Example usage	Will be applied in use case 3 to detect emotive texts.
Comments	• Emotion detection will be applied to Twitter tweets as text corpora. • The format of text annotation needs to be determined. • The service may use ontologies such as sentiWordNet. • The Emotion Detection Service will eventually use the output of the Named Entity Recognition Service and the Sentiment Analysis Service.
Conformance classes	Not yet available.
Implementation rules	Not yet available.
Implementation status	Not yet implemented
UML model	Not yet available.

2.2.5 Relation Extraction

An important step for understanding the semantic content of text is the extraction of semantic relations between entities in natural language documents. Automatic extraction techniques have to be able to identify different versions of the same relation which usually may be expressed in a great variety of ways. Therefore, these techniques benefit from taking into account many syntactic and semantic features, especially parse trees generated by automatic sentence parsers. Typed dependency parse trees are edge and node labelled parse trees whose labels and topology contains valuable semantic clues. This information can be exploited for relation extraction by the use of kernels over structured data for classification.

The service envisaged would extend current dependency tree kernels by including richer structural features, for instance using ordering properties as well as the labelling of nodes in dependency trees in a novel fashion to create kernels that consider most of the available information in dependency trees. To allow the usage of more substructure properties while maintaining an acceptable runtime, the service will be based on two new computation algorithms tailored for relation extraction tree kernels being developed at FHG.

Name	Relation Extraction
Standards	No relevant standards available
Description	The Dicode Relation Extraction Service detects certain prototypical relations between individuals, products, and brands that are related to emotions. The Relation Extraction Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *RelationExtraction*: provides the relation extraction functionality
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*RelationExtraction*
annotate	Provides annotated texts Inputs are text corpora and relation type(s) t be detected.
Example usage	Will be applied in use case 3 to detect certain standard relations
Comments	• Relation extraction will be applied to Twitter tweets as text corpora. • The format of text annotation needs to be determined. • The service may use ontologies such as sentiWordNet. • The Relation Extraction Service will eventually use the output of the Named Entity Recognition, Sentiment Analysis, and Emotion Detection Services.
Conformance classes	Not yet available.
Implementation rules	Not yet available.
Implementation status	Not yet implemented
UML model	Not yet available.

2.3 Subgroup Discovery

Subgroup discovery is a technique for learning descriptive rules, i.e. rules that can be used to understand inherent relations in the data of a database. A subgroup is a subset of individuals in the database such that the individuals in the subgroup are distinguished from all other individuals by their characteristics with regard to some target attribute. Typically, these characteristics ensure that a subgroup displays a different (statistical) distribution on the target attribute if compared to the distribution on the target attribute in the complete dataset. Subgroups are presented in terms of "subgroup patterns", i.e. a conjunction of atomic propositions, the most common being pairs $a = v$ with a being an attribute of the database and v being a value. Then a subgroup pattern of the form $a_0 = v_0, \ldots, a_{n-1} = v_{n-1}$ states that the combi-

nation of values v_i of the attributes a_i are "interesting" with regard to a specific attribute studied. In case of numerical values, atomic propositions may be intervals $l \leq a \leq u$ with l being a lower and u being an upper bound. An instance d_i of the database d is said to satisfy a subgroup pattern p if it satisfies all the atomic propositions in the pattern, e.g.. if $a = v$ is in p, then $d_i(a) = v$ meaning that the value $d_i(a)$ of the instance d_i at attribute has value v.

Subgroup discovery intends to find subgroup patterns that are "interesting" and "easy to interpret". Interestingness is typically expressed in terms of a "quality function" that typically reflects statistical or other user-defined criteria. Moreover, subgroup patterns found are often relatively simple and thus easy to understand. For example, if biomedical experts try to identify which genes may play a role in the development of a disease, gene expressions of each patient's DNA can be related to the relevant clinical data, with particular clinical data being used as target attribute.

Subgroup discovery is a generic name for a variety of algorithms with specific characteristics, for instance

- Type of values of target attribute
  - Nominal
  - Numeric
  - Ordinal
- Subgroup pattern
  - Nominal - $a = v$
  - Intervals - $l \leq a \leq u$. The intervals may be disjoint or overlapping. There are several strategies to define intervals for specific data, e.g. divide data into intervals that hold equal number of values or split the range of all values of an attribute into equal length intervals.
- Quality function
  - Two class quality functions – the values of the target attribute are split into two classes in terms of which the quality function is defined.
  - Multiclass quality functions – the quality function depends on all values of the target attribute.
- Subgroup property
  - Refined subgroups – only maximal elements with regard to a given order on subgroups are considered for the output
  - Closed subgroups – close a subgroup pattern under all patterns that have the same "instance basis", i.e. which occur in all the instances in which the given subgroup pattern occurs.

Each particular choice of algorithm reflects particular aspects of "interestingness" and results in quite different subgroup patterns being generated. Hence working on a use case, subgroup discovery implies exploration of the "space of algorithm". Thus, the choice of algorithms becomes a parameter for subgroup discovery. Other parameters include the length of subgroups to be considered (which may have a dramatic effect concerning computation time and space), number of best subgroups to be considered, minimal quality, and generality.

Because of high number of possible choices, the Dicode Subgroup Discovery Service at present is defined to be a generic service with regard to subgroup discovery algorithms, in that it takes as input the description of a particular (instrumented) subgroup algorithm (instrumented means that parameters of the algorithms may be set). The other parameters comprise

length of subgroups, the number of best subgroups to be considered, minimal quality, and generality.

Name	*Subgroup Discovery Service*
Standards	.csv (http://www.ietf.org/rfc/rfc4180.txt) and .arff (http://www.cs.waikato.ac.nz/~ml/weka/arff.html)
Description	The Dicode Subgroup Discovery Service considers a given target variable and aims to detect novel, potentially useful, and ultimately interesting knowledge given by subgroup patterns with regard to this property of interest. The Subgroup Discovery Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *Discovery*: generates the subgroup patterns. • *NotificationProducer*: allows subscription of notification consumers.
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*Discovery*
generateSubgroups	Input as defined above plus a database in .csv or .arff format. Output is a list of subgroup patterns.
Interface	*NotificationProducer*
subscribe	A subscriber can register given his/her interest as notification consumer to receive notifications. Parameter is callback of the notify operation of the respective notification consumer.
getCurrent-Message	The subscriber pulls the current notification message.
Example usage	The discovery interface will be mainly applied in Use Case 1 – clinico-genomic research for detecting genomic patterns with regard to target taken from the clinical data.
Comments	• The present algorithm uses simple interestingness functions. Future versions will enhance the basic algorithm by more sophisticated interestingness functions and more complex discretisation methods for numeric and ordinal values. • The present algorithm is a variant of the well-known FP-Growth algorithms, but uses only an one-level FP-tree, reusing the structure for recursion. This results in constant space usage and, somewhat unforeseen, reduces computation time in many cases by order of an magnitude if compared with the reference implementation.
Conformance classes	Not yet applicable
Implementation rules	Not yet available.
Implementation	Prototypical version of the algorithm is implemented. Not yet provided

status	as service. For a snapshot of output data see Appendix B.5.
UML model	Not yet available.

2.4 Recommendation and Similarity Learning

2.4.1 Recommender Service

Recommender systems attempt to recommend information items that are likely to be of interest to the user. "Typically, a recommender system compares a user profile to some reference characteristics, and seeks to predict the 'rating' or 'preference' that a user would give to an item they had not yet considered."[24] Many algorithms used for recommender systems and gene pattern recognition are based on distance measures. The distance indicates the similarity of information items. Then, those items are recommended that are "closest" to match the user profile.

The Dicode Recommender Service will be based on models generated by similarity learning. These models depend on the type of information items considered in Dicode. Information items to be considered at present comprise user profiles, documents, and data sets. For every type, one or more specific models will be generated using the Dicode Similarity Learning Service.

User interaction will be used as feedback to improve the similarity models, in that the Dicode system will observe the choices made by the user with regard to the "similar items" offered, extending the list of similar items as input for similarity learning.

The Dicode Recommender Service is generic, in that it will be parameterised by the type of information items, the similarity model, and output options.

Name	*Recommender Service*
Standards	No relevant standards available
Description	The Dicode Recommender Service allows querying the Dicode system for information items that are similar to those provided as input. The search is based on similarity models learned by using the Dicode Similarity Learning Service (see Section 2.4.2). The Recommender Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *Recommendation*: provides query and feedback operations.
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Examples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*Recommendation*
query	Defines a query. Parameters include the type of information items, the simi-

[24] Bell, R. M., Koren, Y., & Volinsky, C. (2007). The BellKor solution to the netflix prize. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.9009

	larity model, and the information item(s) for which similar items to be recommended. An optional integer parameter restricts the number of recommended items. The default is that all similar items are listed.
feedback	Provides a list of information item pairs considered to be similar by the user)
Example usage	A user may want to find similar data sets similar, for instance of clinical trials, or want to find users with a similar profile as the own profile. Using the Dicode Similarity Learning Service, similarity models will be generated within Dicode for • User profiles • Data sets • Patient records. The characteristics of the specific information item types to be used will be specified in cooperation with the other work packages and be included in Deliverable 4.2.2.
Comments	• Since at present detailed specification of information types are not available yet, similarity models for the video lecture completion for recommender systems (http://tunedit.org/challenge/VLNetChallenge) have been generated as a proof-of-concept. • Computation times depend more or less linearly on the number of attributes (hence basic distance measures, see Section 2.4.2) and on the size of the data collection. • Improvement of results depend on improvement of similarity learning.
Conformance classes	Not yet applicable
Implementation rules	Not yet available.
Implementation status	Prototypical version of the algorithm is implemented. Not yet provided as service. For a snapshot of output data see Appendix B.6
UML model	Not yet available.

2.4.2 Similarity Learning Service

The key idea of similarity learning is to replace fixed distance functions by learning a function that produces a non-negative real number for any pair of examples. The intended semantic is that the higher this number the more similar the two examples are. The training data that the function learns from consist of example pairs labelled as similar or dissimilar.

The learning framework is generic, in that information items are considered as structured objects (of arbitrary nature). For instance, the information item may be a document with a substructure given by "title", "author", "abstract", "main text", and, eventually, "metadata". Typically, similarity learning proceeds by attaching basic distance measures to the atomic components, eventually as well to structural properties, and then by learning weight coefficients applied to the basic distance measures. More formally, let D be a set of base distance measures. Then the distance of two "points" x, y is defined by $distance(x,y) = \sum_{d \in D} w_d \times d(x,y)$ where w_d are the weight coefficients. Points are typically n-tuples of atomic components but may as well be structures such as trees or graphs.

Basic distance measures are distinguished by the type of the atomic component. A number of distance measures, well known from the literature [25], are listed below:

- String[26]
  LevensteinSimilarity, BlockDistance, DiceSimilarity, JaroWinklerSimilarity, MatchingCoefficient, JaccardSimilarity, CosineSimilarity on words, ChapmanLengthDeviation, ChapmanMatchingSoundex, ChapmanMatchingSoundexSpanish, Jaro, MongeElkan, NeedlemanWunch, OverlapCoefficient, QGramsDistance, SmithWaterman, SmithWatermanGotoh, SmithWatermanGotohWindowedAffine, SoundexEnglish, SoundexSpanish,
- Numbers, Number series
  EuclideanDistance, CosineSimilarity, CamberraDistance, ChebychevDistance, CorrelationSimilarity, JaccardSimilarity, ManhattanDistance
- Boolean
  Jaccard Similarity, Dice Similarity, Matching Koeffizient, Cosinus Similarity
- Text
  Cosine distance on n-grams

Hierarchy may be reflected by taking the path length into account.

Hierarchy in structures such as trees or graphs may be reflected by taking the path length into account.

The result of similarity learning is a *similarity model* that consists of the weight coefficients for the basic distance measures (i.e. $[w_{d'} \mid d' \in D]$. for an example, see Appendix B.7).

The Dicode Similarity Learning Service is generic, in that it will be parameterised based on the particular distance measures attached to the components of a structured object.

Name	*Similarity Learning Service*
Standards	No relevant standards available
Description	The Dicode Similarity Learning Service allows generating similarity models for structured information. The input consists of the type of the information item, distance measuring functions attached to specific components of the structured information and of two list of information item pairs, one specifying items considered to be similar and the other considered to be dissimilar. The output is a similarity model for the specific type of information items. The Similarity Learning Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *Learner*: provides the learning operation.
Interface	*ServiceCapabilities*
getCapabilities	Informs the requestor about the common and specific capabilities. Exam-

[25] http://reference.wolfram.com/mathematica/guide/DistanceAndSimilarityMeasures.html
[26] http://secondstring.sourceforge.net/doc/iiweb03.pdf

	ples of specific capabilities are the supported information item types, similarity models related to the type, and output options
Interface	*Learner*
computeModel	Takes input as specified above and outputs a similarity model. The operation supports asynchronous interaction.
Interface	*NotificationProducer*
subscribe	A subscriber can register the interest as notification consumer to receive notifications. Parameter is callback of the notify operation of the respective notification consumer.
getCurrent-Message	The subscriber pulls the current notification message.
Example usage	This service will be used for learning similarity of • user profiles • data sets • patient records. The generated models will be used as input for the Dicode Recommender Service (see Deliverable D4.2.1). The characteristics of the specific information item types to be used will be specified in cooperation with the other work packages and be included in Deliverable D4.2.2
Comments	• Computation times depend on the number of attributes/basic distance measures used and on the number of information item pairs labelled as similar and dissimilar. In general, learning needs time in the domain of seconds. • Improvement of results depends very much on the choice of appropriate attributes for a given notion of information item. • One may experiment with a combination of the learning algorithms implemented so far as well as design new base distance measures.
Conformance classes	Not yet applicable
Implementation rules	Not yet applicable.
Implementation status	Prototypical version of the algorithm is implemented. Not yet provided as service. For a snapshot of output data, see Appendix B.7
UML model	References to related UML models if available.

3 External Services

3.1 Introduction

External services comprise services / servers from external sources that will be provided within Dicode Framework. These services considerably extend the scope of Dicode providing access to rich algorithmic sources adding to the flexibility of the framework. Since external documentation is available, no specific description of the services is included.

3.2 Rapidminer Services

The functionality of Rapidminer can be provided as web services via the community edition of RapidAnalytics (http://rapid-i.com/content/view/182/192/lang,en/). Only the Similarity Learning and Recommender Service are at present provided as Rapidminer Service. However, RapidAnalytics provides access to many, possibly complex data mining workflows potentially of use for Dicode.

3.3 R Services

Rserve[27] is a server that allows evaluating R scripts. It listens for any incoming connections and processes incoming requests. Rserve can be obtained as a CRAN package. It provides interfaces to Java[28] and C++. Rserve is used by Taverna (see 3.4) for evaluation of R blocks. Since many algorithms of use for Use Case 1 are available as R scripts, provision of Rserve in the context of Dicode enhances the computational basis of the project. None of the services specified above are available as R scripts.

3.4 Taverna Services

Taverna[29] is a workflow system that allows running scientific workflows. Taverna workflows may include data mining services. Taverna can be deployed in a server version.[30] Workflows can be submitted and monitored. The Taverna Server offers a REST[31] and a SOAP[32] API. Details about the available operations can be found at the respective URLs.

Taverna may be used in Dicode to specify more complex workflows using the services specified above eventually combined R scripts.

[27] http://www.rforge.net/Rserve/doc.html
[28] http://rosuda.org/Rserve/dist/JRclient/JavaDoc/
[29] http://www.taverna.org.uk/
[30] http://www.taverna.org.uk/documentation/taverna-2-x/server/2-2/
[31] http://www.taverna.org.uk/documentation/taverna-2-x/server/2-2/rest-api/
[32] http://www.taverna.org.uk/documentation/taverna-2-x/server/2-2/soap-api/

4 Tentative Implementation Rules for Data Mining Services

The service descriptions above are abstract, presenting only the essential features of the services. Implementation rules give hints how to implement the abstract description – or rather an UML model if available. The rules below are tentative, in that they need practical evaluation.

At present, the intention is to provide Dicode Data Mining Services as RESTful services using the operators GET, PUT, POST, and DELETE.

All data mining services except for the recommendation services will probably share the following characteristics:

- Interaction will be asynchronous (even if the asynchronous interaction interface is not (yet) explicitly specified) since computation times may vary to a great deal so that synchronous interaction would be contra-productive blocking service client and service provider for an unforeseeable amount of time.
- Frequent access to input data (sets) is likely with input data being of considerable size. Hence, input data and computational engine should in general reside on the same host engine.
- Several experiments will take place with regard to a given data set but with varying parameters.
- Operators will be instrumented in that particular parameters will determine their behaviour. As a terminology, we say that the parameters specify a "task".

Deriving from these characteristics, a number of resources can be identified:

Resources	**HTTP Method**	**Description**
`http://path/to/service/`		Endpoint of a service
`http://path/to/service/REST/`		REST API
`http://path/to/service/REST/`	`GET`	Returns the service capabilities (implements getCapabilities)
`…/REST/interface/`	`GET`	Returns the interface capabilities
`…/REST/operation`	`GET`	Alternative to the above if an interface only consists of one operation and its name is not ambiguous.
`…/REST/interface/operation`	`GET`	Show description of the operator
`…/REST/operation/`	`POST`	Start execution of task
`…/REST/operation/`	`GET`	Show description of the operator
`…/REST/operation/`	`DELETE`	Stop the current task
`…/REST/operation/task`	`PUT`	Set the task
`…/REST/operation/task`	`GET`	Show the description of the task
`…/REST/operation/input`	`GET`	Show the required format for input data
`…/REST/operation/input`	`PUT`	Set the input data
`…/REST/operation/result`	`GET`	Return the result/output
`…/REST/operation/status`	`GET`	Show status (calculating, completed or idle; amount of memory)
`…/REST/operation/progress`	`GET`	Return an estimate of the progress of the task
`…/REST/operation/reset`	`DELETE`	Kill task (if exists), clear all caches (input/task)

Some comments are appropriate:

- The "REST" takes care of eventually provided SOAP services.
- Tasks may be specified as parameters to the POST operator or as configuration file in case of more complex instrumentations. The latter is stored under `/task`.
- Data sets as input can be stored as resource under `/input`.
- Instead of storing output data under `/return`, an URL may be provided indicating the from where the resource can be obtained using a get.

The other resources should be self-explaining.

Appendix A Service Interfaces

Subsequently, a list of generic interfaces is proposed that should be used by Dicode services if applicable to promote some uniformity for the sake of interoperability.

A.1 Service Capabilities Interface

The Service Capabilities interface allows uniform querying for the properties of a service, so that a potential user of the service gets information about what can be expected in terms of interfaces, operations, and data types. It is proposed that the Service Capabilities interface is mandatory for all Dicode services.

Name	*Service Capabilities Interface*
Standards	The *getCapabilities* operation of the Service Capabilities Interface is backward compatible with the concepts and definitions of the *getCapabilities* operation as defined in OGC 05-008c1 Web Services Common Specification V1.0.[33]
Description	The Service Capabilities Interface defines a uniform way to get a self-description of a Service in terms of service meta-information (used for, e.g., purposes such as service discovery and service invocation). The *ServiceCapabilities* interface is a mandatory interface to be implemented by all Dicode Services.
Interface	*ServiceCapabilities*
getCapabilities	Informs about the capabilities of a service instance. In addition to capabilities common to all Dicode services, a Dicode service may provide a specific set of capabilities. Furthermore, different meta-information schemas supported. The meta-information schema shall be structured according to the rules for Dicode application schemas.
Example usage	The Service Capabilities Interface contributes to a consistent description of the functionality provided by Dicode Services.
Comments	An example of service meta-information will be provided as UML model.
Conformance classes	Not defined
Implementation rules	Not yet available
UML model	Not yet available

A.2Asynchronous Interaction

The Asynchronous Interaction interface supports asynchronous calls of operation.

Name	*Asynchronous Interaction Interface*
Standards	• W3C Web Services Addressing V1.0 Core, http://www.w3.org/TR/ws-addr-core/ The concept of an End-

[33] http://www.opengeospatial.org/standards/common

	point Reference (EPR) as defined by WS-Addressing can be used for the invokeAsync operation in order to specify the entity to which notifications are to be sent as a result of asynchronous operation execution. • OASIS Web Services Notification (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn)
Description	A service providing the Asynchronous Interaction Interface shall indicate in its service capabilities which operations can be executed by means of the invokeAsync operation. For this purpose, the general description of an operation (which is part of the common service capabilities) shall be extended for the invokeAsync operation. The extension consists of a list of the operations that are supported by means of the invokeAsync operation. This includes also the input and output parameters of each such operation.
Interface	*AsynchronousInteraction*
invokeAsync	The mandatory invokeAsync operation starts asynchronous execution of an operation. The invokeAsync operation returns immediately returning an identifier to the caller (invocation ID) that represents the asynchronous execution. In order to allow the executing service instance to deliver notifications, an entity must be referenced in the call of invokeAsync that provides a callback interface. The service instance calls the notify operation of that entity to deliver notifications resulting from asynchronous operation execution. This may be done using the subscribe operation of the NotificationProducerInterface. The callback may refer to the notify operation of the NotificationConsumer interface. If no such entity is referenced, notifications will not be delivered which may be sufficient in certain cases. Any exception thrown by the invokeAsync operation signals that the asynchronous execution has failed to start. Especially if the OperationRequest parameter contains an invalid operation name, the invokeAsync operation throws an InvalidParameterValue exception. If the operation name is valid but asynchronous execution of that operation is not supported, the invokeAsync operation throws an AsyncOperationUnsupported exception. Any exception thrown by the asynchronously executing operation does not lead to an exception of the invokeAsync operation but is delivered to the provider of the callback interface.
abort	The mandatory abort operation aborts execution of a previously invoked asynchronous operation identified by its invocation ID which was returned when calling the invokeAsync operation. An exception indicates that it is not possible to abort the operation. If the operation has terminated in the meantime (by itself or due to a previous abort) the abort operation may either throw an InvalidParameterValue exception or an InvalidState exception. This is implementation-dependent: If the information that the operation has already terminated is still available, the abort operation may throw an InvalidState exception. Otherwise, an InvalidParameterValue exception indicates that the invocation ID is no longer referring to an existing operation invocation.

Example usage	Asynchronous interaction is typically used for operations that do not return a result synchronously, for instance due to long computation or transmission time
Comments	An example of service meta-information will be provided as UML model.
Conformance classes	Not defined
Implementation rules	Not yet available
UML model	Not yet available

A.3 Notification

Notification comprises a number of interfaces supporting notification between services.

Name	*Notification Interface*
Standards	Web Services Base Notification 1.3 (WS-BaseNotification), OASIS Standard, 1 October 2006[34]
Description	OASIS WS-Notification is a family of related specifications that define a standard Web services approach to notification using a topic-based publish/subscribe pattern. A Web service (NotificationProducer) disseminates information (notifications) to a set of other Web services (NotificationConsumers) each of which has been subscribed to the producer previously.
Interface	*NotificationConsumer*
notify	A NotificationProducer may produce a notify message containing one or more notifications.
Interface	*NotificationProducer*
subscribe	A subscriber can register the interest as notification consumer to receive notifications. Parameter is callback of the notify operation of the respective notification consumer.
getCurrentMessage	The subscriber pulls the current notification message.
Interface	*PullPoinInterface*
getMessages	If a requestor wishes to retrieve Notification Messages accumulated by the PullPoint, it sends a getMessages request to the PullPoint endpoint.
destroyPullPoint	The PullPoint interface provides a destroy operation, providing a means by which a requestor can terminate the PullPoint resource.
Interface	*CreatePullpoint*
createPullPoint	If a requestor wishes to create a new PullPoint resource, it MUST send a createPullPoint request 1066 to an endpoint supporting the PullPoint interface.
Interface	*SubscriptionManager*
renew	To modify the current lifetime of a Subscription, a requestor sends a renew request message to the SubscriptionManager

[34] http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

unsubscribe	To terminate a Subscription, a requestor sends an unsubscribe request message to the SubscriptionManager.
pauseSubscription	To temporarily suspend the production of Notifications on the given Subscription. a requestor sends an pauseSubscription request message to the SubscriptionManager.
resumeSubscription	If a requestor wishes to resume the production of Notifications on the given Subscription, it must send a resumeSubscription request message
Example usage	Given a long running data mining service, alert may indicate termination or additional storage requirements or other similar measures.
Comments	
Conformance classes	Not defined
Implementation rules	Not yet available
UML model	Not yet available

Appendix B Service Output

For each service that is implemented as a prototype, example input and output values are provided.

B.1 Twitter Harvester

Name	Twitter Harvester
Input	A scan operation which selects the result for one arbitrary key in the HBase column store is performed via the HBase shell[35] using the following command: hbase(main):001:0> scan 'thtweets', LIMIT => 1
Output	The cell content for all columns are printed to the shell as shown in the following example: ROW COLUMN+CELL \x00\x00\x00\x00\x00\x00\x00\x14 column=d:creationDate, timestamp=1314006046327, value=\x00\x00\x01\x0A\x1E\x92op \x00\x00\x00\x00\x00\x00\x00\x14 column=d:from, timestamp=1314006046327, value=Jack Dorsey \x00\x00\x00\x00\x00\x00\x00\x14 column=d:fromId, timestamp=1314006046327, value=\x00\x00\x00\x00\x00\x00\x00\x0C \x00\x00\x00\x00\x00\x00\x00\x14 column=d:hashtags, timestamp=1312756962468, value= \x00\x00\x00\x00\x00\x00\x00\x14 column=d:inReplyToStatusId, timestamp=1312756962468, value=-1 \x00\x00\x00\x00\x00\x00\x00\x14 column=d:inReplyToUserId, timestamp=1312756962468, value=-1 \x00\x00\x00\x00\x00\x00\x00\x14 column=d:lang, timestamp=1314006046327, value=en \x00\x00\x00\x00\x00\x00\x00\x14 column=d:retweetCount, timestamp=1314006046327, value=\x00\x00\x00\x00\x00\x00\x00e \x00\x00\x00\x00\x00\x00\x00\x14 column=d:source, timestamp=1314006046327, value=web \x00\x00\x00\x00\x00\x00\x00\x14 column=d:text, timestamp=1314006046327, value=just setting up my twttr \x00\x00\x00\x00\x00\x00\x00\x14 column=d:to, timestamp=1312756962468, value=-1 1 row(s) in 0.3280 seconds
Note	The REST interface for Twitter Harvester is not available yet as stated before. The example shows a query to the HBase column store used by Twitter Harvester. As a client the HBase shell is used.

[35] http://hbase.apache.org/shell.html

B.2 Twitter Pre-processing

Name	Twitter Pre-processing
Input	http://path/to/service/REST/twitterpreprocessing/twittervectorexport/tweets?q=/(\s)(Apple|APPLE|apple)(\s)&minSupport=3
Output	Dictionary: Key class: class org.apache.hadoop.io.Text Value Class: class org.apache.hadoop.io.IntWritable Key: 03: Value: 0 Key: 4tab.me: Value: 1 Key: 8g: Value: 2 Key: acabei: Value: 3 Key: adventures: Value: 4 Key: ahhhhh: Value: 5 Key: along: Value: 6 Key: anders: Value: 7 Key: aren't: Value: 8 Key: bed: Value: 9 Key: bezig: Value: 10 Key: bill: Value: 11 Key: blunt: Value: 12 Key: bra: Value: 13 Key: bukan: Value: 14 Key: cabin: Value: 15 Key: cali: Value: 16 Key: callin: Value: 17 Key: cheer: Value: 18 Key: compote: Value: 19 Key: confirms: Value: 20 Key: creative: Value: 21 Term-vectors: Key class: class org.apache.hadoop.io.Text Value Class: class org.apache.mahout.math.VectorWritable Key: 48847207671148544: Value: {960:1.0,958:1.0,709:1.0,328:1.0,575:1.0,1409:1.0,1096:1.0,1373:1.0,504:1.0} Key: 48865791407423489: Value: {73:1.0,960:2.0,1329:1.0,1066:1.0,225:1.0,886:1.0,60:1.0,355:1.0,95:1.0,905:1.0,679:1.0,522:1.0,1409:1.0,1000:1.0} Key: 48871106358153216: Value: {669:1.0,198:1.0,1000:1.0,213:1.0,930:1.0,259:1.0,726:1.0,55:1.0,1260:1.0,522:1.0,653:2.0,384:1.0,716:1.0,246:1.0,1183:1.0,1316:1.0,1004:1.0,167:1.0,1234:1.0,1377:1.0,736:1.0,894:1.0,960:1.0,682:1.0,1146:1.0,942:1.0,204:1.0,1409:1.0}

B.3 Keytrends Service

Name	Keytrends Service
Input	http://path/to/service/REST/keytrends/twittermetadata/hashtags?q=apple&date=20110623[36]

Output	{
	"result": [
	{
	"#apple": 5193
	},
	{
	"#iphone": 1283
	},
	{
	"#rt": 1066
	},
	{
	"#ipad": 918
	},
	{
	"#ff": 412
	},
	{
	"#mac": 303
	},
	{
	"#ipad2": 277
	},
	{
	"#wheniwas13": 261
	},
	{
	"#ipod": 251
	}
	...
	]
	}

B.4 Topic Detection

Name	Topic Detection
Input	http://${server}:${port}/REST/topic_detection/
Output	0 0,00433 ada haha aku aja cont mau itu apa dan 1 0,00276 tco ipad details tech free iphone new apply inside 2 0,01008 que para los por iphone tco del las una

[36] Please note that the Url differs from the abstract service description. As typical for a RESTful resource the Url does not contain a method name like *getTweets*, but addresses the resource directly.

	3 0,00324 die und iphone der das ich für mit ist 4 0,00517 ios iphone ipad beta jailbreak android tco news video 5 0,01406 iphone ipad tco アプリ □□□□ mac w□□ android applestore 6 0,00209 time iphone что ipad tco capsule techland ios для 7 0,00677 juice pineapple drink orange express lol like green ciroc 8 0,00453 snapple pineapples pineapple juice good apples eating love lol 9 0,00187 blackberry just fruits imessage life easier bbm damnitstrue det 10 0,00172 bite eve dear did stupid ddlovato angry birds sauce 11 0,00492 music cloud icloud service cut final pro google tco 12 0,0041 store sqcom i'm applebee's new big appleton ave card 13 0,00159 para ipad macbook sorteio quem applebrasll iphones tweet seguindo 14 0,00291 les pour sur des tco que pas d'apple iphone 15 0,01121 pineapple apples pie juice chicken cake good just cinnamon 16 0,0019 tnwto thenextweb tnwapple ios mpanzarino apple's mtt tco safari 17 0,00576 ipod case touch ipad iphone generation amznto skin cover 18 0,03238 just store new iphone like ipad i'm don't tco 19 0,00277 iphone tco рублей акция tumblrcom photo всего heyapples акция 20 0,00687 iphone ipad new september white tco release launch apple's 21 0,00292 win ipad giveaway tco follow chance followers iphone retweet 22 0,0041 itunesapplecom album app itunes single new buy music feat 23 0,00596 android nokia patent iphone google rim mobile apple's smart-phone 24 0,00277 ipad tablet wifi tco amznto モデ□ iphone model dlvrit 25 0,00291 iphone app free ipad download appstore user air visit 26 0,00399 iphone new ipad unlocked brand black free white buy 27 0,00197 follow icreate_id info magazine creative lifestyle magz icreate generation 28 0,00793 big thanks follow love tco happy live ヲ□□ new 29 0,00093 strength money aids sex power girls cont death deckly 30 0,00192 hey tinyurlcom free iphone ipho love ipad like did 31 0,00135 tree far fall apples like say guys doesn't best 32 0,00217 pie jeans boots fur want cream berry ice pies 33 0,00547 iphone location tracking data tco google patent users iphones 34 0,00123 spongebob pineapple dude pineapples drunk sea lives know wasn't 35 0,00129 day away keeps doctor cute fruit 아이폰 allkpop appler 36 0,00366 ipad free copy paste tinyurlcom here's getfreeipadtwo claim giving 37 0,00217 iphone ipod charger usb amznto cable applewd car adapter 38 0,00378 mac lion malware update tco security defender software thunderbolt 39 0,0016 just ipad スキンシー□ free bityme iphone got □□□□ cool 40 0,0301 lol like just i'm applebees got know lmao don't 41 0,00317 cider vinegar apples today tco daily weight stories health 42 0,00228 pineapple mango smoothie mcdonalds say remember just instapaper dropbox

	43 0,00678 apples like eat love oranges just pineapple i'm green 44 0,00419 ipad apple's sales tco aapl foxconn explosion earnings quarter 45 0,00237 bad apples tco nicoms red マイ □□□ like nicovideo badapple 46 0,01547 applebees i'm store day just big good going applebee's 47 0,00286 tco apples new eye list sonyapple anonymous news law 48 0,00633 icloud wwdc ios lion apple's tco keynote jobs steve 49 0,00252 iphone tco ipad twitter ipod mac social makes store 50 0,00173 youtube video iphone lebron edition james vibrates rings tco 51 0,00467 macbook pro air laptop inch imac mac core new 52 0,00284 een van met voor het dat ipad niet iphone 53 0,00646 app store apps tco ipad apple's mac amazon ios 54 0,00676 que com não mais pra iphone para por uma 55 0,00492 jobs steve retail new tco apple's store stores campus 56 0,00385 samsung patent iphone ipad galaxy sues tco lawsuit files 57 0,00147 quer applebrasll ipad tweet dar basta nesse macbook ganhar 58 0,00396 google microsoft facebook twitter tco intel ios amazon worth 59 0,002 pineapple express imac_noapple logo windows teamfollowback

B.5 Subgroup Mining

Name	Subgroup Mining
Input	nqfpgrowth.native –conf " All_ER.conf"
Output	TargetValueDistribution: total= 48 values= 42~positive / 6~null Contains 2 subgroups. 1. Hs.79136 == (0.05 - 0.55) Quality = 0.0797307453518 Size = 116, Generality = 0.382838283828 p = 0.931034482759 2. Hs.149923 == (-0.05 - 0.05) Quality = 0.0687187530634 Size = 122, Generality = 0.402640264026 p = 0.893442622951 3. Hs.26770 == (-0.05 - 0.05) & Hs.170328 == (-0.05 - 0.05) Quality = 0.0680325458288 Size = 96, Generality = 0.316831683168 p = 0.9375 4. Hs.1519 == (-0.05 - 0.05) & Hs.79136 == (0.05 - 0.55) Quality = 0.0669215436395 Size = 102, Generality = 0.336633663366 p = 0.921568627451 5. Hs.169946 == (-0.05 - 0.05) Quality = 0.0668888671045 Size = 120, Generality = 0.39603960396 p = 0.891666666667

	Considered 434513535 subgroups with depth <= 4 , out of 1945885146812076 with maxDepth 4 making 2.23298654451e-05% in a tree built of 14701 terms
Note	The REST interface for Subgroup Mining is not available yet. The example shows an application call using a configuration file. The format of the configuration file is JSON. It will be used as parameter for the RESTful service.

B.6 Recommender

Name	Similarity Learning
Input	*Attributes of video lecture metadata* id, type, parent_id, language, name, description, slide_titles, type_unpopularity, authors, author_categries, categories, event_language, event_parent_id, event_name, event_descr, texts, author_unpopularity, allCategories,textAndCategories *Example of video lecture metadata* 24, 'lecture', 14, 'sl', "'Od ognja do sladoleda - kemijski eksperimenti'", "'Predavanje; predstavitev eksperimentov'", NULL, 0.964992317392647, NULL, NULL, 'Chemistry', 'sl', 1, "'RTK Tekmovanje IJS v znanju raÄunalniÅ¡tva 2006 - Ljubljana'", "'Tekmovanje IJS v znanju raÄunalniÅ¡tva je namenjeno dijakinjam in dijakom", "'Od ognja do sladoleda - kemijski eksperimenti' NULL 'Predavanje; predstavitev eksperimentov'", NULL, 'Chemistry', "'Od ognja do sladoleda - kemijski eksperimenti' NULL 'Predavanje; predstavitev eksperimentov' 'Chemistry'"
Output	"ID_1";"Recommendation" 24.0;"9316.0, 9318.0, 9319.0, 9320.0, 9321.0, 9317.0, 9314.0, 9104.0, 9313.0, 9315.0"
Note	The learned model relates to the ECML PKDD Challenge http://www.ecmlpkdd2011.org/challenge.php Cold start Problem. A model has been learned. Input is a video lecture (identified by id). Output is a list of identifiers of „nearest“ video lectures ordered by increasing distance.

B.7 Similarity Learning

Name	Similarity Learning
Input	Consists of lists of pairs of
Output	CosineSimilarity_texts: w = 0.07924750606723298 ScoreDistance_author_unpopularity: w = 0.1537865723349271 DiceSimilarity_language: w = 0.08074249501291282 CosineSimilarity_tags: w = 0.18415741922101114 CosineSimilarity_event_name: w = 0.0812718930807026 CosineSimilarity_event_descr: w = 0.07460593119192621 DiceSimilarity_event_parent_id: w = 0.057610195582139

	HierarchicalDistance_categories: w = 0.05951105323471455 HierarchicalDistance_parent_id: w = 0.0638039637727783 CosineSimilarity_parent_id_parents:w = 0.05499192052382489
Note	The learned model relates to the ECML PKDD Challenge http://www.ecmlpkdd2011.org/challenge.php Cold start Problem. A model has been learned. Input is a video lecture (identified by id). Output is a list of „nearest“ video lectures ordered by increasing distance.