

# KNOWLEDGE DISCOVERY USING THE KNOWMINER FRAMEWORK

Werner Klieber, Vedran Sabol, Markus Muhr,  
*Know-Center Graz*  
*Inffeldgasse 21a, Austria*

Roman Kern, Georg Öttl, Michael Granitzer  
*Know-Center Graz*  
*Inffeldgasse 21a, Austria*

## ABSTRACT

Services for increasing information quality in unstructured information are central to mostly all applications of information management. Service-oriented computing is a promising paradigm for embedding such intelligent services in different application scenarios and heterogeneous IT landscapes. In this paper we present our knowledge discovery framework named “KnowMiner”, a service oriented architecture designed with the primary goal to ease usage for non knowledge discovery experts while providing a rich set of knowledge discovery functionalities for very different application scenarios. We will introduce the underlying data model of the framework, its basic services as well as concepts for workflow composition based on pre- and post-conditions. We summarize our findings on the development and application of such a framework in a set of design aspects and outline that formal pre- and post-conditions are not suited for data intensive applications with a high spectrum of functionality.

## KEYWORDS

Knowledge Discovery, Service oriented Architectures, Design Aspects

## 1. INTRODUCTION

“Data is the next Intel inside” is a mission statement of a large number of innovative Web 2.0 applications. It stresses the importance of intelligent handling of data and information for future applications. Nowadays, well established information retrieval systems organize information in indices or tables. Users need to interpret the results and extract needed knowledge on their own. Knowledge discovery focuses on the semi-automatic identification of previously unknown, potential new knowledge from unstructured information [Fayyad et. al. 1996]. It is an interdisciplinary research area including concepts from information retrieval, machine learning, logic/inference and visualization [Weiss et. al. 2004].

Although integrating methods from these areas into a consistent framework is highly desirable it imposes a set of different technical and organizational challenges: Usually the development team has to consist of experts from different fields (e.g. information visualization, logics, machine learning etc.), capable to implement and combine sophisticated, highly specialized algorithms. While providing powerful methods, the framework must be well documented, easy usable for application developers and has to provide intelligent debugging utilities for very different application scenarios. Without in depth knowledge on algorithms, application developers must be empowered to develop different workflows for particular application scenarios.

Nowadays service oriented architectures (SOA) provide a natural approach for solving this dilemma between complex functionality and easy applications: decoupled services, developed by independent service providers are used to assemble complex applications. The vision is to provide an easy consumption of services by being independent from technical implementations [Elfatraty 2007]. Ongoing research in standards such as the Web Service Modelling Ontology (WSMO) [Dumitru 2006] focuses further on automating service consumption by enriching the services with additional semantic information. WSMO

builds on a goal oriented approach using ontologies to translate user's intentions to service instances. Mediators are used to compensate for semantic differences between the services and for mapping the different data formats.

However, as stated in [Mogul 2006], emergent misbehavior will occur in systems when the complexity of the system increases. Low-level interactions between services as well as the number of independent services have profound implications on the overall system. This reduces promised SOA benefits of reducing the complexity by being independent of individual service behavior. Integrating additional logic to automate processes may result in further emergent malfunctions and increased coupling of services. Jeffrey Mogul comes to the conclusion, that new approaches are needed and we never will have tools to solve all misbehavior, but that should not stop us from trying.

In this paper we present our knowledge discovery framework named "KnowMiner", a service oriented architecture designed with the primary goal to ease usage for non knowledge discovery experts while providing a rich set of knowledge discovery functionalities for very different application scenarios. The supported knowledge discovery functionality is basically limited by two factors: (i) the data model and (ii) the number of different services; aspects described in section 2 and 3. While we are not going into too much detail, we provide a comprehensive overview for grasping the idea on the complexity in managing data intensive knowledge discovery workflows. Especially composition and debugging methods of those workflows are usually time consuming, error prone tasks. We try to ease those tasks by defining pre- and post-conditions based on the defined data model as well as a clear service architecture (see section 4).

The ambitious goal to ease the usage for non knowledge discovery experts required several development cycles. Our experience in very different project - supplemented by questionnaire based evaluations in each development cycle - revealed underlying design guidelines, especially on the service level. Section 5 summarizes the essence of those guidelines and our findings gathered during the last 5 years.

Therein, the main contribution of this work is twofold: Firstly, we identify essential building blocks for knowledge discovery frameworks and outline important design aspects for their composition. In contrast to previous work which mainly addresses particular aspects we try to combine all different aspects for such a framework and derive underlying design guidelines. To give to prominent examples in [Raspl 2004] standards for exchanging data mining and knowledge discovery models are outlined. However, aspects of service modularization and workflow composition are completely ignored. On the other hand [Ferruci & Lalla 2004] describe the components of IBM's Unstructured Information Management Architecture (UIMA). Although workflows and their application are outlined, details on the different service components as well as the definition of pre- and post-conditions are missing. Also, more complex knowledge discovery tasks like visualization and clustering are not outlined.

As a second contribution of this paper we work out our experience in developing different applications on top of the framework as well as the viewpoint and experience of non-knowledge discovery experts in solving knowledge discovery tasks. Outlining pitfalls and success factors may help other research and development groups in developing or applying similar frameworks.

## 2. DATA MODEL

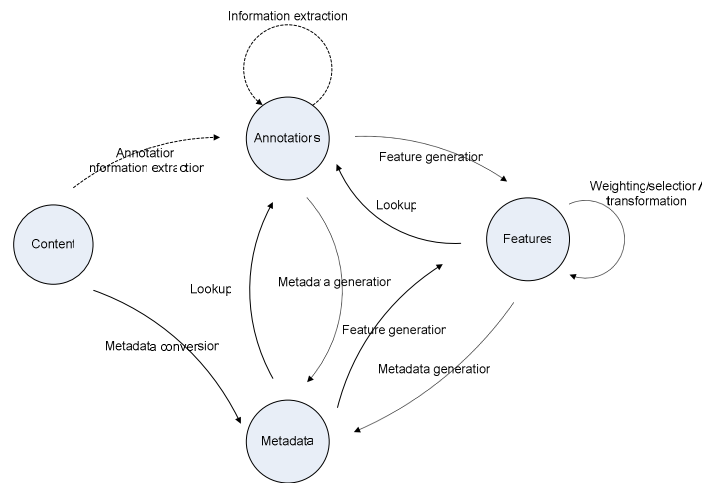
Knowledge discovery involves data driven processes. Data needs to be transformed and processed by algorithms to extract meaningful knowledge for humans. In our model the basic objects to work with are information entities [Granitzer 2006] which can be seen as information container having assigned different properties. The initial data filled into an information entity is the content, and during the execution of workflows algorithms will manipulate the properties of those information entities, create new entities or identify relationships between those entities. Particular properties of information entities are:

- Metadata is additional data assigned to an information entity either converted from the data source the entity is read from (e.g. file name or a creation date of an image) or through some metadata generating algorithm (e.g. a text classifier automatically assigning a document to a particular class or not)
- Annotations are created by information extraction methods which add further information to an entity. For example given the textual content of an information entity, a NLP pipeline may add

word class (noun, adverb) information to a single word or may identify that a set of words represent a person or some other kind of named entities.

- Features represent the statistical information for most knowledge discovery algorithms. For example a large number of machine learning methods require real valued vectors in a so called feature space (see [Weiss et. al. 2004] for details). Features are generated either by estimating statistic over selected content parts with particular annotations (e.g. counting the frequency of nouns) or by converting different metadata (e.g. creation data).

Figure 1 depicts the different properties and their conversion. Also, to give a concrete example consider a set of simple text documents. Each of these documents is an information entity and has assigned metadata like file size. Tokenization and Part-of-Speech Tagging annotate each character sequence with its function (e.g. word or punctuation) and its word class (e.g. noun). All nouns are then used to create the features of a vector space which is then subsequently used for clustering documents to topical related groups.



**Figure 1 Transformation within information entities. Full lines identify information reducing transformations, plain lines indicates information preserving transformations. Slashed lines indicate information enriching transformations.**

Also, information entities are embedded in some structure, like files structured in the file systems. RDF - the Resource Description Framework – has been used to provide such structural elements, which also differentiates the KnowMiner Framework from other existing frameworks like UIMA. However, we found that for most of today’s knowledge discovery tasks complex relationships can be safely ignored. Moreover, the management of such structures increases the complexity of algorithmic development as well as decreases the applicability for application developers.

In order to keep the access to properties and structural elements of information entities simple but yet powerful, a common data access had been realized. The common data access allows to define different, application scenario dependent views on the data. In our approach each data aspect is named by a unique string based identifier. Adapters implement the logic to map those data aspect on a set of properties of information entities. Each key can be bound to a specific data structure helping developers dealing with the data. Introducing such data abstraction allows for changing data representation in the background without affecting its external behavior, a key ingredient for developers in larger teams. Domain specific views can be easily added by replacing and modifying the adapters for domain specific needs. Addressing data with single keys enables an intuitive and flexible way specifying the input and output of services. Using alias names for identifiers allows developers compensate differences between similar projects. This simplifies reusing configurations in other projects. To give an example, information extraction annotates text with different named entities like persons, organizations and locations. For a particular application scenario we are only interested in all named entities, but not in particular classes of named entities. By providing an access key for named entities and a mapping logic application developers can access those named entities by a invoking a single call.

### 3. BASIC SERVICES

Workflows in service oriented architectures consist of single, atomic services, called basis services. For our domain we analyzed various application scenarios and decomposed knowledge discovery tasks into a set of basic services. Our goal was to keep the number of services low in order to reduce the complexity in workflow orchestration as well as to reduce the number of services a developer is responsible for. We split knowledge discovery topics into single basic services, each manageable by a team of 1-3 persons. The interface of each service has to fulfill certain criteria like for example that its operation signatures have to be independent of the underlying backend and algorithmic details. Hence, the service interface has to be compatible with a processing interface specifying a common way for using a service: setting input data and retrieving results, starting and stopping service execution, etc.

**Import Service:** Mostly the first step in each project is to gather required data from foreign data sources and format the data into an internal Meta format that allows for easy access to all data aspects within the environment. This service transforms external data into information entities resp. their properties which is then made accessible to other services by the above mentioned common data access.

**Information Extraction Service:** This service annotates content of single information entities. It extracts named entities like persons, places or organizations from text content. Information extraction makes implicit information stored in unstructured text computer processable. Information extraction results provide additional faceted dimensions for search results and can be used to discover additional relationships between information entities. Processing is done by sending a document through a pipe of processing algorithms and providing the generated annotations within our common data structure.

**Feature Extraction Service:** The task of this service is to generate meaningful features from annotations and metadata. It supports different common weighting and feature selection schemes. Features can be real valued, nominal or boolean and are managed via a set of vectors.

**Information Retrieval Services:** For users an important task in projects is to search for information. Information retrieval tasks are well understood, fast and accepted by users. Mostly the results are shown in lists ranked by relevance to a query. For each result some metadata is provided along a content overview, such as a snippet. Faceted search approaches add additional orthogonal aspects to search results like presenting persons or geographic locations mentioned within these results [Efthimiadis 1996]. This provides users with multiple views on the search result set. In our framework an index service is used to store the content of information entities in a structure appropriate for searching. The service allows specifying which content aspects are stored in which searchable fields and how they can be queried (for instance in a case sensitive or insensitive manner).

**Association service:** An association service is used retrieve to similar terms to a given term. These similar terms can be used during query specification to provide keyword suggestions to users, e. g. for query expansion. Search fields can be put into relation by using their document term distribution [Reginald et al. 1995]. For instance, having a search field for countries and one for cities allows searching for a country and retrieving associated cities.

**Summarization Service:** This service extracts metadata from a set of information entities by examining the coherence between a set of entities. For each entity some descriptive summarizations can be generated. An example for such a summarization service is a keyword extraction service. For each entity keywords are extracted describing its content as good as possible in respect to the given set. Firstly, for each document a feature vector is generated. These vectors are weighted using a TF/IDF algorithm. The term frequency (TF) indicating how often a term occurs in a document is divided by the amount of documents this term occurs in. This favours terms that occur frequently in the examined document and weakens terms that occur frequently in many documents since they have not much information entropy. This avoids having the same keywords for all entities.

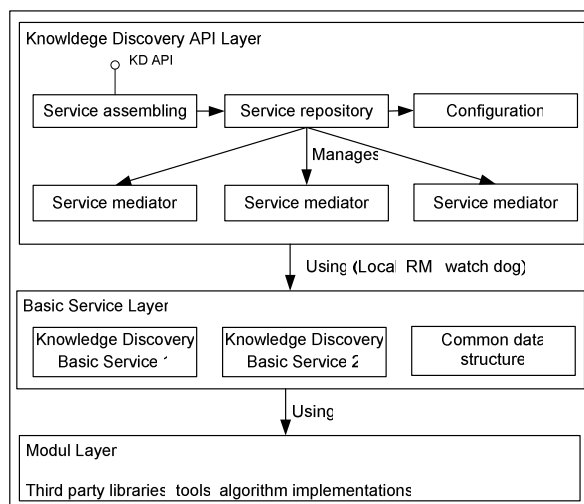
**Clustering-Service:** Clustering service identifies groups of related information entities (such as documents) which are represented by high-dimensional vectors. Relatedness between any pair of entities is expressed by computing the distance (or similarity coefficient) between the corresponding vectors. Typically Euclidean distance or cosine similarity is employed, although other coefficients, such as dice or jaccard, may also be used. The service encapsulates a variety of clustering algorithms such as k-means, ISODATA algorithm, hierarchical agglomerative clustering, affinity propagation, BIRCH and some others [Xu, Rui and Wunsch, D., 2005]. Additionally, the clustering service provides means for projecting the clustered data set

into a low dimensional space (typically 2D screen space) in such a way that high-dimensional distances are preserved (as far as possible). In the resulting layout similar objects are placed close while dissimilar ones are positioned far from each other. The resulting low-dimensional coordinates of projected entities are used by visualization components designed for visual analysis and interactive exploration of large, high-dimensional data sets [Andrews et al, 2002]. As of 2008 the service has been successfully applied on large, high-dimensional corpora containing up to one million text documents.

**Classification Service:** Classification is a supervised learning approach for assigning documents to a given set of concepts. For example classification can be used to distinguish between spam and no spam mail. Blog or wiki documents can be assigned automatically to categories. Firstly the classifier is trained on a training document set containing information to which concepts the documents belong to. Afterwards new documents can be assigned automatically to the concepts. The classifier uses features from the features service as input and returns the concept association suggestions together with a confidence value. The classifier service wraps several classification methods like k-Nearest Neighbors, support vector machines, boosting based approaches, naive bayes und rocchio classifier.

#### 4. SERVICE COMPOSITION

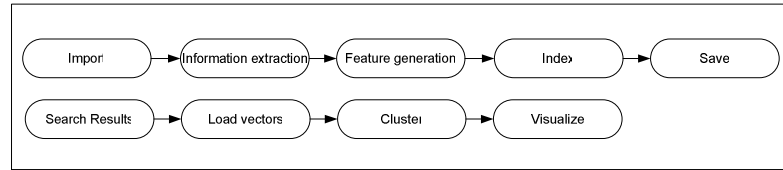
Based on the defined data model and the introduced basic services we designed a service oriented architecture as depicted in Figure 2. However, although having a common data structure and a common interface concept, assembling of these services was still too complicated for application developers. Extra work needs to be done preparing the data in the common data structures to meet service requirements. Several semantic inconsistencies between service constraints that each team has introduced in their implementation need to be adjusted. So a further abstraction layer is introduced with the goal to encapsulate the main workflow as well as to distribute services among different nodes using Java RMI. We designed an own coarse grained knowledge discovery interface with focus on consistent data types where the output of an operation can be directly used as input for the next operation. To fill the gap between this interface and the basic services, mediator services are implemented resolving semantic discrepancies between the services. Also, a watch dog has been assigned to critical services allowing for terminating those services in case operation lasts extraordinary long. This allows us to keep some degree of fail-safety.



**Figure 2 SOA based system architecture showing the service level breking down high level knowledge discovery API to low level modules.**

Applications use the knowledge discovery interface by invoking a series of operation calls to assemble workflows. Figure 3 shows common knowledge discovery task. The first workflow shows a typical preprocessing task. The second workflow shows how to cluster search results. The index generated in the

preprocessing workflow is used to search for interesting information. Next the saved features are loaded into the information entities of the search results and clustering is performed. Inspecting the most relevant terms of each cluster allows users to discover the main topics covered by the search result.



**Figure 3 Illustration of a common knowledge discovery task. The first workflow shows the required preprocessing steps to cluster search results in the second workflow.**

We introduced a simple pre/post condition mechanism to help developers finding out the correct order the operations need to be executed. To gain first experience we started with a simple approach. We used the data aspects described in section 2 – content, annotations, metadata and features. Each service specifies what kind of data it needs as input (pre condition) and what data it will write (post condition). In a first check we used the preconditions before executing a service to ensure the needed data is available. This should help application developers to assemble correct workflows and to reduce the amount of debugging workflows. Another aspect of those pre- and postconditions was to formally specify the requirements of a service through the service developer thereby documenting the intended application. Application developers should get an idea on which services may be required to run a-priori of a specific service. However, it turned out that information about missing properties of information entity is not useful for developers. Without any further information they do not know how to correct the problem. Providing useful help needs additional context information about assembled workflows and the current task the developer wants to perform within a workflow. For instance, a developer needed to cluster search results. When executing clustering, he got an error that features are missing. So he executed the feature generating service beforehand. However, the pre conditions for the feature extraction service were not fulfilled at this stage because no content is available for search results.

Developers of the basic services implemented this pre- post conditions information sparsely. They considered it as additional work with no direct benefit worth doing it. For some services it is also impossible to provide this simple information immediately. They can only provide this conditions information when additional data is available: runtime settings, configuration parameters or the task to perform. While our model on pre- and postconditions could be semantically enriched to cover those aspects, this yields to an dramatic increase in modeling efforts. So it is unlikely that developers would engage in such efforts resulting in a decrease in development efficiency. So introducing further semantics for a richer, more complex description of pre- and post-conditions seems to be unfeasible.

## 5. DESIGN ASPECTS

Our design goal was in building an environment for applying frequently used knowledge discovery tasks in different projects and to provide knowledge discovery functionality for a broad range of application scenarios. The framework itself has been developed over several years with a team of approximately 8-10 persons and has been applied to roughly 8 different projects. Such application scenarios includes providing intelligent knowledge services for workplace integrated learning [Moedritscher et. al. 2007], context detection for knowledge workers [Granitzer et. al. 2008], visualizing unstructured documents [Sabot et. al. 2008] and multi modal retrieval of meeting information [Sabot et. al. 2007]. Thus, very different teams of application developers integrated the functionality of the framework in different application scenarios.

Clearly, supporting a broad spectrum of functionality while easing application development for non-experts are contradicting requirements. We found that the granularity of services in the KnowMiner Framework as well as the provision of the common data access supports both requirements. Especially the common data access pattern allowed for separating the application oriented viewpoint on the underlying data

view from the knowledge discovery viewpoint. Service granularity not only supported application developers, but also simplified the organization of service development teams by reducing the number of services a team member is responsible for.

One central point for knowledge discovery applications is that they are mostly data driven with the additional downside that the data is unstructured. Unstructured data requires high fail-safety, since data can be very noisy. Noisy data often yields to unpredictable algorithmic behavior, resulting in concrete exceptional states at its best and non-termination at its worse. Ideally, one could validate workflows a priori whether a correct execution is possible or not. Also, due to the massive amount of data usually processed means for monitoring the quality of each processing step have to be established. For example, errors in extracting the text out of different document formats may not directly lead to an error during indexing those documents, but may reduce the quality of subsequent processing like document clustering or searching for those documents. This combination of noise and massive amount of data also drives the required time for identifying error, especially for the application developer. With the introduction of pre- and post-conditions we tried to address those aspects: application developers should get a better understanding on why a workflow fails and service developers should be able to clearly specify the transformational function of their services. However, it turned out that this formal specification received very low acceptance on both sides and did not improve service composition and debugging capabilities of workflows.

A certain degree of fail-safety could be achieved through the watch dog concept in combination with distributing services over RMI. Fail-safety and the independence of workflow/service execution on different nodes were highly appreciated by application developers. This is especially true since data intensive workflows require large pools of system resources like main memory which can be more efficiently managed in a distributed manner.

Generalizing from our architecture, we found that the following properties are essential for an efficient application development in the domain of knowledge discovery:

- **Reduced data complexity:** Supporting a broad spectrum of application scenarios also requires to support a broad spectrum of information objects like for example textual artifacts, multimedia artifacts etc. However, for application developers this data complexity must be hidden somehow.
- **Reduced number of service (methods):** A larger number of service and/or service methods results in more combination possibilities and a shallower learning curve on the application developer side.
- **Quality of Service parameters:** QoS parameters for knowledge discovery applications are hard to define, since algorithmic complexity as well as quality depends strongly on the data processed. For example documents consisting of several thousand pages are hardly processable by some preprocessing algorithms.
- **Guidance in and quality of workflow composition:** Although we kept the number of services and service methods small, application developer require guidance in workflow composition. Especially since applications are data intensive, testing and debugging takes more time than in non data driven domains. Due to the unstructured nature of the data we do not see that formal semantics in terms of pre- and post-condition may help to increase workflow composition, at least without an extra ordinate amount of additional work.
- **Debugging capabilities for workflow execution and status of the data model.** In practice, errors during workflow execution occur most often after already processing a significant amount of data items and causal relationships are not always clear. For example clustering of documents strongly relies on the preprocessing. However, preprocessing can be executed long time before, with different configurations and thus if clustering fails one also has to validate the previously occurring step of preprocessing. Again, a time consuming aspect.

## 6. CONCLUSION

In this paper we presented our knowledge discovery framework and our design approach for provide an easy usage for developers. We outlined the framework including data model, basic services and service architecture. We introduction a pre post condition mechanism intended to enable a more failsafe and transparent consumption of the functionality. From our experience in the application of the project and questionnaires undertaken during development, we derived a set of basic design aspects. Besides those

design aspects our main finding regards the introduction of pre- and post-conditions for knowledge discovery services: providing even very simple semantic enhancements lead to a formidable implementation complexity. The simple approach didn't fulfill expected prospects and a exhaustive semantic infrastructure is needed to benefit (maybe) from semantic descriptions. For real world applications this extra effort may far beyond project resources.

For our goal using classical approaches like documentation and sample source code is a simpler way explaining the framework concept to developers. Documentations have the problem of staying up to date with technical changes. Currently we investigate in tools to extract useful concepts and workflows automatically from source code and in applying Wikipedia to create communities around the developers.

## ACKNOWLEDGEMENT

The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

## REFERENCES

- Andrews K. et al, 2002. The InfoSky Visual Explorer: Exploiting Hierarchical Structure and Document Similarities. *Information Visualization*, Vol. 1, No. 3-4, pp 166-181.
- Dumitru Roman, Jos de Bruijn, 2006, WWW: WSMO, WSML, and WSMX in a Nutshell, in *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 516-522.
- Efthimiadis E. N., 1996, Query Expansion, *Annual Review of Information Systems and Technology*, Vol. 31, 121--187.
- Elfatary, A., 2007, Dealing with change: components versus services, *Commun. ACM* 50(8), 35--39.
- Fayyad, U. M.; Piatetsky-Shapiro, G. & Smyth, P. From Data Mining to Knowledge Discovery: An Overview. *Advances in Knowledge Discovery and Data Mining*, 1996, 1-34
- Ferrucci, D. & Lally, A. (2004), 'Building an example application with the Unstructured Information Management Architecture', *IBM Systems Journal, Special Issue on Unstructured Information Management* 43, p. 455-476.
- Granitzer M., 2006, Konzeption und Entwicklung eines generischen Wissenserschliessungsframeworks, PhD thesis, Know-Center, Austria.
- Granitzer, M.; Kröll, M.; Seifert, C.; Rath, A. S.; Weber, N.; Dietzel, O. & Lindstaedt, S., 2008, Analysis of Machine Learning Techniques for Context Extraction *Proceedings of 2008 IEEE International Conference on Digital Information Management (ICDIM08)*
- Mogul, J. C., 2006, Emergent (mis)behavior vs. complex software systems, in *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, ACM, New York, NY, USA, pp. 293—304
- Moedritscher, F.; Hoffmann, R.; Klieber, W., "Integration and Semantic Enrichment of Explicit Knowledge through a Multimedia, Multi-source, Metadata-based Knowledge Artefact Repository", *I-Know '07*, Graz, Austria, 5-7 September 2007
- Raspl, S. (2004), 'PMML Version 3.0 - Overview and Status' *KDD-2004 Workshop on Data Mining Standards, Services and Platforms (DM-SSP 04)*, *KDD-2004 The Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Reginald Ferber et al., 1995, An Associative Model of Word Selection in the Generation of Search Queries, in *Journal of the American Society for Information Science*, 46(9), pp. 685-699.
- Sabol, V.; Andrews, K.; Kienreich, W. & Granitzer, M. Text mapping: Visualising unstructured, structured, and time-based text collections *Intelligent Decision Technologies*, 2008, 2, 117-128
- Sabol, V.; Gütl, C.; Neidhart, T.; Juffinger, A.; Klieber, W. & Granitzer, M. (2007), Visualization Metaphors for Multimodal Meeting Data, in *'Datenbanksysteme in Business, Technologie und Web (BTW 2007)*, Workshop Proceedings, 5.-6. März 2007, Aachen, Germany', pp. 250-269.
- Weiss, S.; Indurkha, N.; Zhang, T. & Damerou, F. *Text Mining: Predictive Methods for Analyzing Unstructured Information* Springer, 2004
- Xu, Rui and Wunsch, D., 2005. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, Vol. 16, No. 3, 645-678.