Ersch. in: ACM SIGKDD Explorations Newsletter ; 11 (2009), 1. - S. 26-31

http://dx.doi.org/10.1145/1656274.1656280

KNIME – The Konstanz Information Miner

Version 2.0 and Beyond

Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel and Bernd Wiswedel
University of Konstanz
Nycomed Chair for Bioinformatics and Information Mining
Box 712, 78457 Konstanz, Germany
Michael.Berthold@Uni-Konstanz.DE

ABSTRACT

The Konstanz Information Miner is a modular environment, which enables easy visual assembly and interactive execution of a data pipeline. It is designed as a teaching, research and collaboration platform, which enables simple integration of new algorithms and tools as well as data manipulation or visualization methods in the form of new modules or nodes. In this paper we describe some of the design aspects of the underlying architecture, briefly sketch how new nodes can be incorporated, and highlight some of the new features of version 2.0.

1. INTRODUCTION

The need for modular data analysis environments has increased dramatically over the past years. In order to make use of the vast variety of data analysis methods around, it is essential that such an environment is easy and intuitive to use, allows for quick and interactive changes to the analysis process and enables the user to visually explore the results. To meet these challenges data pipelining environments have gathered incredible momentum over the past years. Some of today's well-established (but unfortunately also commercial) data pipelining tools are InforSense KDE [6], Insightful Miner [7], Pipeline Pilot [8], to name just three examples. These environments allow the user to visually assemble and adapt the analysis flow from standardized building blocks, which are then connected through pipes carrying data or models. An additional advantage of these systems is the intuitive, graphical way to document what has been done.

KNIME, the Konstanz Information Miner provides such a pipelining environment. Figure 1 shows a screenshot of the standard KNIME workbench with a small example data analysis workflow. In the center, a flow is reading in data from two sources and processes it in several, parallel analysis flows, consisting of preprocessing, modeling, and visualization nodes. On the left a repository of nodes is shown. From this large variety of nodes, one can select data sources, data preprocessing steps, model building algorithms, as well as visualization tools and drag them onto the workbench, where they can be connected to other nodes. The ability to have all views interact graphically (*visual brushing*) creates a powerful environment to visually explore the data sets at hand. KNIME is written in Java and its graphical workflow editor is implemented as an Eclipse [9] plug-in. It is easy to extend through an open API and a data abstraction framework, which allows for new nodes to be quickly added in a well-defined way.

In this paper – which is based on an earlier publication [1] concentrating on KNIME 1.3 – we describe the internals of KNIME in more detail with emphasis on the new features in KNIME 2.0. More information as well as downloads can be found at `http://www.knime.org`. Experimental extensions are made available at the KNIME Labs pages (`http://labs.knime.org`).

2. OVERVIEW

In KNIME, the user can model workflows, which consist of nodes that process data, transported via connections between those nodes. A flow usually starts with a node that reads in data from some data source, which are usually text files, but databases can also be queried by special nodes. Imported data is stored in an internal table-based format consisting of columns with a certain (extendable) data type (integer, string, image, molecule, etc.) and an arbitrary number of rows conforming to the column specifications. These data tables are sent along the connections to other nodes that modify, transform, model, or visualize the data. Modifications can include handling of missing values, filtering of column or rows, oversampling, partitioning of the table into training and test data and many other operators. Following these preparatory steps, predictive models with machine learning or data mining algorithms such as decision trees, Naive Bayes classifiers or support vector machines are built. For inspecting the results of an analysis workflow numerous view nodes are available, which display the data or the trained models in diverse ways.

In contrast to many other workflow or pipelining tools, nodes in KNIME first process the entire input table before the results are forwarded to successor nodes. The advantages are that each node stores its results permanently and thus workflow execution can easily be stopped at any node and resumed later on. Intermediate results can be inspected at any time and new nodes can be inserted and may use already created data without preceding nodes having to be re-executed. The data tables are stored together with the workflow structure and the nodes' settings.

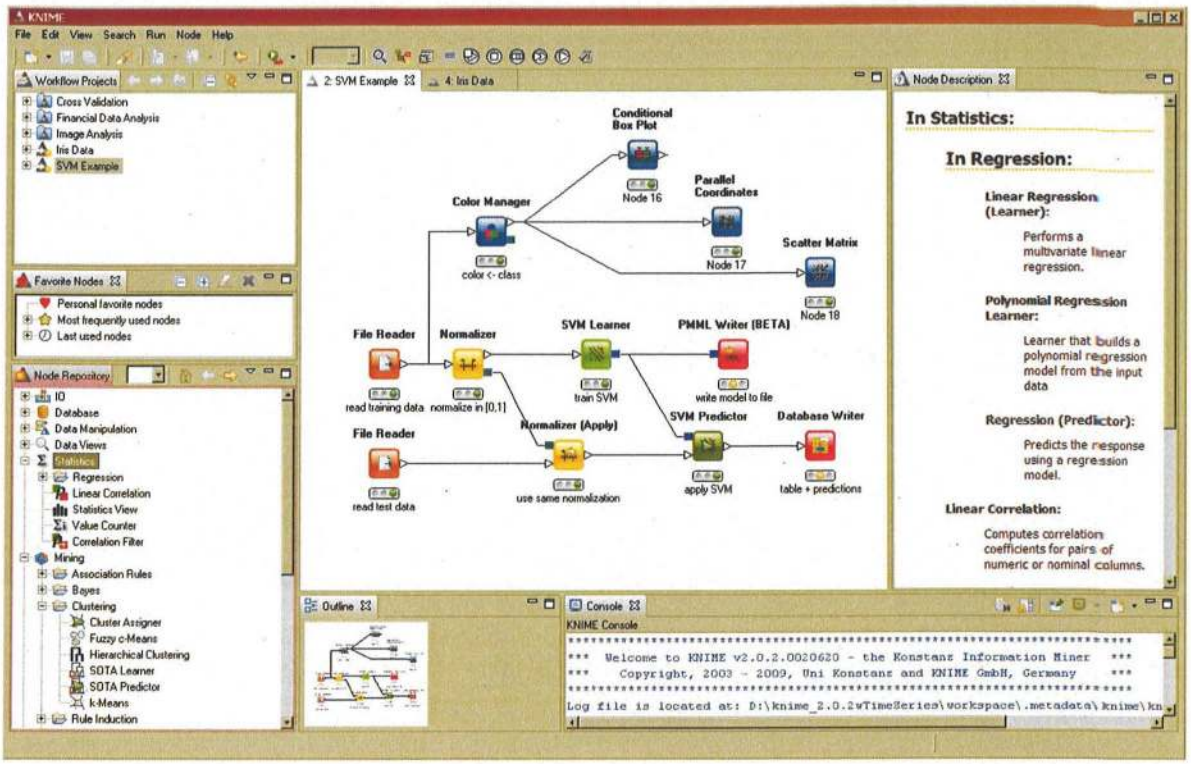

Figure 1: The KNIME workbench with a small example workflow.

One of KNIME's key features is *hiliting*. In its simplest form, it allows the user to select and highlight several rows in a data table and the same rows are also highlighted in all other views that show the same data table (or at least the highlited rows). This type of hiliting is simply accomplished by using the 1:1 correspondence between the tables' unique row keys. However, there are several nodes that change the input table's structure and yet there is still some relation between input and output rows. A good example for such a 1-to-n relation are clustering algorithms. One of the node's input are the training (or test) patterns, the output are cluster prototypes. Each of the clusters covers several input patterns. By hiliting one or more clusters in the output table all input patterns which are part of those cluster(s) are hilited in the input table. Similar translations are, of course, also possible for other summarizing models: branches/leaves of a decision tree, frequent patterns, discriminative molecular fragments, to name just three examples.

One of the important design decisions was to ensure easy extensibility, so that other users can add functionality, usually in the form of new nodes (and sometimes also data types). This has already been done by several commercial vendors but also by other university groups or open source programmers. The usage of Eclipse as the core platform means that contributing nodes in the form of plugins is a very simple procedure. The official KNIME website offers several extension plugins for business intelligence and reporting via BIRT [2], statistical analysis with R[4] or extended machine learning capabilities from Weka [5], among many others.

3. ARCHITECTURE

The architecture of KNIME was designed with three main principles in mind.

- Visual, interactive framework: Data flows should be combined by simple drag&drop from a variety of processing units. Customized applications can be modeled through individual data pipelines.
- Modularity: Processing units and data containers should not depend on each other in order to enable easy distribution of computation and allow for independent development of different algorithms. Data types are encapsulated, that is, no types are predefined, new types can easily be added bringing along type specific renderers and comparators. New types can be declared compatible to existing types.
- Easy extensibility: It should be easy to add new processing nodes or views and distribute them through a simple plugin mechanism without the need for complicated install/deinstall procedures.

In order to achieve this, a data analysis process consists of a pipeline of nodes, connected by edges that transport either data or models. Each node processes the arriving data and/or model(s) and produces results on its outputs when requested. Figure 2 schematically illustrates this process. The type of processing ranges from basic data operations such as filtering or merging to simple statistical functions, such as computations of mean, standard deviation or linear

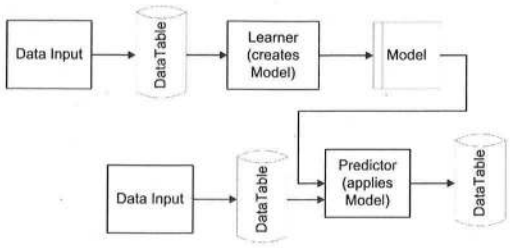

Figure 2: A schematic for the flow of data and models in a KNIME workflow.

regression coefficients to computation intensive data modeling operators (clustering, decision trees, neural networks, to name just a few). In addition, most of the modeling nodes allow for an interactive exploration of their results through accompanying views. In the following we will briefly describe the underlying schemata of data, node, workflow management and how the interactive views communicate.

3.1 Data Structures

All data flowing between nodes is wrapped within a class called `DataTable`, which holds meta-information concerning the type of its columns in addition to the actual data. The data can be accessed by iterating over instances of `DataRow`. Each row contains a unique identifier (or primary key) and a specific number of `DataCell` objects, which hold the actual data. The reason to avoid access by Row ID or index is scalability, that is, the desire to be able to process large amounts of data and therefore not be forced to keep all of the rows in memory for fast random access. KNIME employs a powerful caching strategy which moves parts of a data table to the hard drive if it becomes too large. Figure 3 shows a diagram of the main underlying data structure.

3.2 Nodes

Nodes in KNIME are the most general processing units and usually resemble one node in the visual workflow representation. The class `Node` wraps all functionality and makes use of user defined implementations of a `NodeModel`, possibly a `NodeDialog`, and one or more `NodeView` instances if appropriate. Neither dialog nor view must be implemented if no user settings or views are needed. This schema follows the well-known Model-View-Controller design pattern. In addition, for the input and output connections, each node has a number of `Inport` and `Outport` instances, which can either transport data or models. Figure 4 shows a diagram of this structure.

3.3 Workflows

Workflows in KNIME are essentially graphs connecting nodes, or more formally, a direct acyclic graph (DAG). The workflow manager allows the insertion of new nodes and addition of directed edges (connections) between two nodes. It also keeps track of the status of nodes (configured, executed, ...) and returns, on demand, a pool of executable nodes. This way the surrounding framework can freely distribute the workload among a couple of parallel threads or – as part of the KNIME Grid Support and Server (currently under development) – even a distributed cluster of compute servers. Thanks to the underlying graph structure, the workflow

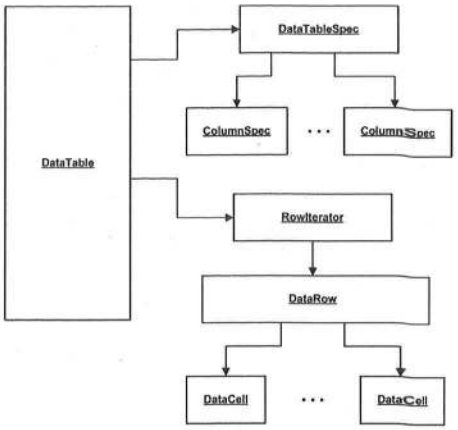

Figure 3: A diagram of the data structure and the main classes it relies on.

manager is able to determine all nodes required to be executed along the paths leading to the node the user actually wants to execute.

3.4 Developing Own Nodes

KNIME already includes plug-ins to incorporate existing data analysis tools. It is usually straightforward to create wrappers for external tools without having to modify these executables themselves. Adding new nodes to KNIME, also for native new operations, is easy. For this, one needs to extend three abstract classes:

- `NodeModel`: this class is responsible for the main computations. It requires to overwrite three main methods: `configure()`, `execute()`, and `reset()`. The first takes the meta information of the input tables and creates the definition of the output specification. The `execute` function performs the actual creation of the output data or models, and `reset` discards all intermediate results.

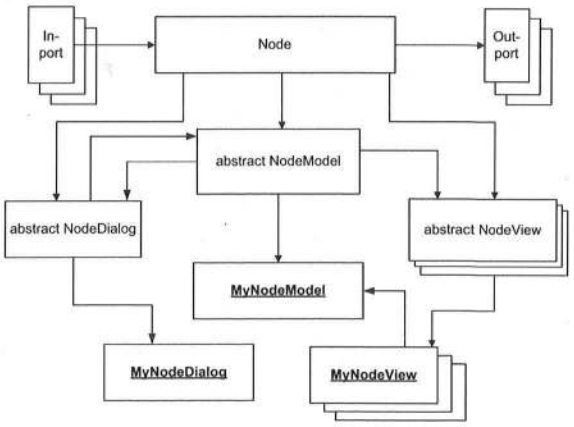

Figure 4: A diagram depicting the main classes of a KNIME node.

- `NodeDialog`: this class is used to specify the dialog that enables the user to adjust individual settings that affect the node's execution. A standardized set of `DefaultDialogComponent` objects allows the node developer to quickly create dialogs when only a few standard settings are needed.
- `NodeView`: this class can be extended multiple times to allow for different views onto the underlying model. Each view is registered with a `HiLiteHandler` which sends events when other views have hilited points and allows launching events in case points have been hilit inside this view.

In addition to the three model, dialog, and view classes the programmer also needs to provide a `NodeFactory`, which serves to create new instances of the above classes. The factory also provides names and other details such as the number of available views or a flag indicating absence or presence of a dialog.
A wizard integrated in the Eclipse-based development environment enables convenient generation of all required class bodies for a new node.

3.5 Views and Interactive Brushing

Each Node can have an arbitrary number of views associated with it. Through receiving events from a so-called `HiLiteHandler` (and sending events to it) it is possible to mark selected points in such a view to enable visual brushing. Views can range from simple table views to more complex views on the underlying data (e. g. scatterplots, parallel coordinates) or the generated model (e. g. decision trees, rules).

3.6 Meta Nodes

So-called *Meta Nodes* wrap a sub workflow into an encapsulating special node. This provides a series of advantages such as enabling the user to design much larger, more complex workflows and the encapsulation of specific actions. Whereas previous KNIME versions had only a fixed set of meta nodes (1 or 2 data input/output ports), it is now possible to create meta nodes with an arbitrary number and even type of ports (see section 5.2) by using a simple wizard. These meta nodes can even be nested and copied. In earlier versions of KNIME also customized meta nodes were available, which allowed for a repeated execution of the enclosed sub workflow, offering the ability to model more complex scenarios such as cross-validation, bagging and boosting, ensemble learning etc. This concept has been replaced by the more powerful loop concept described below (see section 5.1).

3.7 Distributed Processing

Due to the modular architecture it is easy to designate specific nodes to be run on separate machines. But to accommodate the increasing availability of multi-core machines, the support for shared memory parallelism also becomes increasingly important. KNIME offers a unified framework to parallelize data-parallel operations. Sieb et al. (2007) described earlier experiments along those lines, which investigated the distribution of complex tasks such as cross validation on a cluster or a GRID.
In the near future, high performance usage of KNIME will be be supported through a KNIME Grid Engine, which allows distribution of nodes, metanodes, but also chunks of individual node executions on a grid.

4. EXTENSIONS

KNIME already offers a large variety of nodes, among them are nodes for various types of data I/O, manipulation, and transformation, as well as the most commonly used data mining and machine learning algorithms and a number of visualization components. These nodes have been specifically developed for KNIME to enable tight integration with the framework concerning memory policies, progress report and interactive views. A number of other nodes are wrappers, which integrate functionality from third party libraries. In particular, KNIME integrates functionality of several open source projects that essentially cover all major areas of data analysis such as Weka [5] for machine learning and data mining, the R environment [11] for statistical computations and graphics, and JFreeChart [10] for visualization. More application specific integrations allow to make use of the Chemistry Development Kit (CDK [13]) and add molecular data types as well as functionality to compute properties of molecular structures. In the chemoinformatics domain a number of commercial vendors have also integrated their tools into KNIME.
The R integration in KNIME probably offers the most powerful extension, allowing for the execution of R commands in a local R installation or on an R server to build models which can be later used by a R Predictor node. The R view node enables the usage of R views and the R To PMML node allows conversion of a given R object into a corresponding PMML object. In effect, through KNIME it is possible to use essentially all R functionality within an easy to use, intuitive environment for data loading, preprocessing and transformation (ETL).
KNIME 2.0 supports the new Weka version 3.5.6 [5]. Apart from the roughly 50 classifiers that were already part of the Weka-Integration in version 1.3, meta-classifiers, cluster and association rule algorithms have also been integrated adding up to a total of approximately 100 Weka nodes in KNIME. The new Weka port objects (see Section 5.2) are another important new feature in KNIME 2.0. They enable a trained classifier or cluster model to be stored along with the used attributes and the target column. A view on this port lets the user explore the model or clustering that has been built with the training data. This model can be used to predict unseen data with the new Weka predictor node or to assign new data instances to clusters with the Weka cluster assigner node.
The integration of these and other tools not only enriches the functionality available in KNIME but has also proven to be helpful to overcome compatibility limitations when the aim is on using these different libraries in a shared setup.

5. NEW FEATURES IN VERSION 2.0

Besides a number of new nodes and a lot of work under the hood (see the KNIME website at `http://www.knime.org/` for more details), we will discuss the following new features in more detail: support for loops in the workflow, a new concept of user-defined port objects in addition to data tables, improved database connectivity by using the new port objects, and the support of PMML in common data mining algorithms.

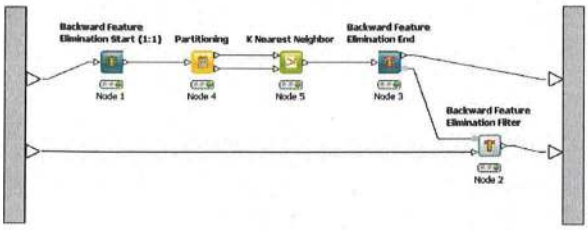

Figure 5: A feature elimination loop inside a meta node that iteratively removes one attribute after another, starting with the worst (i.e. whose removal degrade model quality the least).

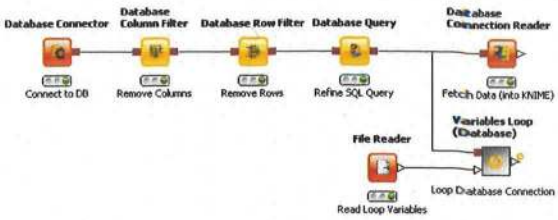

Figure 6: Workflow with nodes that use the new database connections.

5.1 Loop support

The workflows' conceptual structure is a directed acyclic graph, i.e. there are no loops from the output of one node to the input of one of its predecessors. Data flows strictly in one direction. However, there are cases in which the repeated execution of parts of the workflow with changed parameters is desirable. This can range from simple iterations over several input files, to cross validation where a model is repeatedly trained and evaluated with different distinct parts of data, to even more complex tasks such as feature elimination. In order to be able to model such scenarios in KNIME, two special node types have been introduced: loop start- and loop end-nodes. In contrast to normal nodes (inside the loop) they are not reset while the loop executes, each of both nodes has access to its counterpart, and they can directly exchange information. For example, the loop end node can tell the start node which column it should remove at the next iteration or the start node can tell the end node if the current iteration is the last or not. Figure 5 shows a feature elimination loop in which the start and end nodes are visually distinguishable from normal nodes. The feature elimination model can then be used by the feature elimination filter to remove attributes from the data table.
KNIME 2.0 contains several pre-defined loops encapsulated in meta nodes in addition to the individual loop nodes themselves:

- Simple "for" loop, executing a given number of times
- Cross validation
- Iterative feature elimination
- Looping over a list of files
- Looping over a list of parameter settings

Programmers can easily write their own loop nodes by simply implementing an additional interface. Of course, in order to fully make use of the loop-concept it is also necessary to pass variable information to nodes. This allows for e.g. writing out intermediate results to a series of files with a parametrized file name or running a series of experiments with different parameter settings. *Flow Variables* where added in KNIME 2.0 to allow for these types of control parameters. The current implementation is still experimental and will likely be adapted in future versions so we refer to the online documentation for further details concerning this concept.

5.2 Port objects

In previous KNIME versions there were two types of ports, data ports and model ports. The latter did not distinguish between the actual type of models be it a decision tree, a neural net or even color information for data rows used in views. Therefore it was possible to connect a decision tree learner with a neural network predictor and an error was only reported upon execution of the flow. From the programmer's point of view, in certain cases it was quite complicated to translate a model into the used data structure (nested key-value pairs) and back. KNIME 2.0 adds arbitrary port types that can easily be defined by the programmer. This has two main advantages: for the user it is now impossible to connect incompatible ports and the programmer is responsible for (de)serializing the transferred "port object" herself. This is usually much easier than using the old-style method and requires considerably less memory (and space on disk) for big models because the nested hash maps are omitted.

5.3 Improved database support

The new database ports are a prime example of the new port object implementation, Figure 6 shows a small example. These database ports (dark red squares) pass on a connection that encapsulates the parameters used to establish a database connection via a JDBC-compliant bridge.
In the example above, the database nodes work directly in the database by modifying and wrapping SQL statements. The SQL statement itself is only executed when the data is imported into KNIME with the Database Connection Reader node (transition from database to data port). All other nodes, such as Database Connector, Database Column Filter, Database Row Filter and Database Query node perform well-defined operations on the SQL statement. In this example the database connection settings are adjusted within the Connector node and passed to the Database Column Filter and the Row Filter node. The filter nodes offer a user-friendly way to filter out columns and rows without modifying any SQL statement by hand. For advanced users, the SQL query node can be used to manually edit the SQL statement. The outport view for each of those nodes supports a quick look into the database settings, the database meta data and – upon user request – the preview of the current data inside the database.

5.4 PMML

The Predictive Model Markup Language (PMML [3]) is an open standard for storing and exchanging predictive models such as cluster models, regression models, trees or support vector machines in XML format. Ideally, a model trained by KNIME (or any other tool supporting PMML) and stored as PMML can be used in R, SAS Enterprise Miner or, since ver-

sion 2.0, also in KNIME. Almost all basic KNIME nodes that create a model represent it in PMML (if the standard supports it). The corresponding predictor nodes take PMML as input. For PMML exchange between tools, PMML reader and writer nodes have been added as well. However, one should keep in mind that the underlying PMML standard often offers a number of optional attributes in the model, which are usually only understood by the same application that created the model, meaning that in some cases interoperability is limited. One big drawback is currently that the preprocessing is not exported as part of the PMML file, which is a feature that will be addressed in a future version of KNIME.

6. CONCLUSIONS

KNIME, the Konstanz Information Miner offers a modular framework, which provides a graphical workbench for visual assembly and interactive execution of data pipelines. It features a powerful and intuitive user interface, enables easy integration of new modules or nodes, and allows for interactive exploration of analysis results or trained models. In conjunction with the integration of powerful libraries such as the Weka machine learning and the R statistics software, it constitutes a feature rich platform for various data analysis tasks.

New features in KNIME 2.0, especially support for loops, database connection manipulations and PMML further enhance KNIME's capabilities to make it a powerful data exploration and analysis environment with a strong integration backbone that allows for easy access to a number of other data processing and analysis packages.

7. REFERENCES

[1] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel (2007). KNIME: The Konstanz Information Miner. In *Data Analysis, Machine Learning and Applications - Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation e.V.*, Studies in Classification, Data Analysis, and Knowledge Organization, pp. 319–326. Springer, Berlin, Germany.

[2] BIRT. Business Intelligence and Reporting Tools. http://www.eclipse.org/birt/.

[3] Data Mining Group. Predictive Model Markup Language (PMML). http://www.dmg.org/.

[4] R Project. The R Project for Statistical Computing. http://www.r-project.org/.

[5] Ian H. Witten and Eibe Frank (2005). *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, 2nd edition.

[6] Inforsense Ltd.: InforSense KDE. http://www.inforsense.com/kde.html.

[7] Insightful Corporation: Insightful Miner. http://www.insightful.com/products/iminer/default.asp.

[8] SciTegic: Pipeline Pilot. http://www.scitegic.com/products/overview/.

[9] Eclipse Foundation (2008): *Eclipse 3.3 Documentation.* http://www.eclipse.org.

[10] Gilbert, D. (2005): *JFreeChart Developer Guide.* Object Refinery Limited, Berkeley, California. http://www.jfree.org/jfreechart.

[11] R Development Core Team (2007): *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. http://www.R-project.org.

[12] Sieb C., Meinl T., and Berthold, M. R. (2007): Parallel and distributed data pipelining with KNIME. *Mediterranean Journal of Computers and Networks, Special Issue on Data Mining Applications on Supercomputing and Grid Environments.* vol. 3, no. 2, pp. 43-51.

[13] Steinbeck, C., Han, Y. Q., Kuhn, S., Horlacher, O., Luttmann, E., and Willighagen, E.L. (2005): The Chemistry Development Kit (CDK): An open-source Java library for chemo- and bioinformatics. *Journal of Chemical Information and Computer Sciences.* vol. 43, pp. 493-500.