Chapter X
Ontology-Based Construction of Grid Data Mining Workflows

Peter Brezany
University of Vienna, Austria

Ivan Janciak
University of Vienna, Austria

A Min Tjoa
Vienna University of Technology, Austria

ABSTRACT

This chapter introduces an ontology-based framework for automated construction of complex interactive data mining workflows as a means of improving productivity of Grid-enabled data exploration systems. The authors first characterize existing manual and automated workflow composition approaches and then present their solution called GridMiner Assistant (GMA), which addresses the whole life cycle of the knowledge discovery process. GMA is specified in the OWL language and is being developed around a novel data mining ontology, which is based on concepts of industry standards like the predictive model markup language, cross industry standard process for data mining, and Java data mining API. The ontology introduces basic data mining concepts like data mining elements, tasks, services, and so forth. In addition, conceptual and implementation architectures of the framework are presented and its application to an example taken from the medical domain is illustrated. The authors hope that the further research and development of this framework can lead to productivity improvements, which can have significant impact on many real-life spheres. For example, it can be a crucial factor in achievement of scientific discoveries, optimal treatment of patients, productive decision making, cutting costs, and so forth.

INTRODUCTION

Grid computing is emerging as a key enabling infrastructure for a wide range of disciplines in science and engineering. Some of the hot topics in current Grid research include the issues associated with data mining and other analytical processes performed on large-scale data repositories integrated into the Grid. These processes are not implemented as monolithic codes. Instead, the standalone processing phases, implemented as Grid services, are combined to process data and extract knowledge patterns in various ways. They can now be viewed as complex workflows, which are highly interactive and may involve several subprocesses, such as data cleaning, data integration, data selection, modeling (applying a data mining algorithm), and postprocessing the mining results (e.g., visualization). The targeted workflows are often large, both in terms of the number of tasks in a given workflow and in terms of the total execution time. There are many possible choices concerning each process's functionality and parameters as well as the ways a process is combined into the workflow but only some combinations are valid. Moreover, users need to discover Grid resources and analytical services manually and schedule these services directly on the Grid resources essentially composing detailed workflow descriptions by hand. At present, only such a "low-productivity" working model is available to the users of the first generation data mining Grids, like **GridMiner** (Brezany et al., 2004) (a system developed by our research group), DiscoveryNet (Sairafi et al., 2003), and so forth. Productivity improvements can have significant impact on many real-life spheres, for example, it can be a crucial factor in achievement of scientific discoveries, optimal treatment of patients, productive decision making, cutting costs, and so forth. There is a stringent need for automatic or semiautomatic support for constructing valid and efficient data mining workflows on the Grid, and this (long-term) goal is associated with many research challenges.

The objective of this chapter is to present an ontology-based workflow construction framework reflecting the whole life cycle of the knowledge discovery process and explain the scientific rationale behind its design. We first introduce possible workflow composition approaches—we consider two main classes: (1) manual composition used by the current Grid data mining systems, for example, the GridMiner system, and (2) automated composition, which is addressed by our research and presented in this chapter. Then we relate these approaches to the work of others. The kernel part presents the whole framework built-up around a data mining ontology developed by us. This ontology is based on concepts reflecting the terms of several standards, namely, the predictive model markup language, cross industry process for data mining, and Java data mining API. The ontology is specified by means of OWL-S, a Web ontology language for services, and uses some concepts from Weka, a popular open source data mining toolkit. Further, conceptual and implementation architectures of the framework are discussed and illustrated by an application example taken from a medical domain. Based on the analysis of future and emerging trends and associated challenges, we discuss some future research directions followed by brief conclusions.

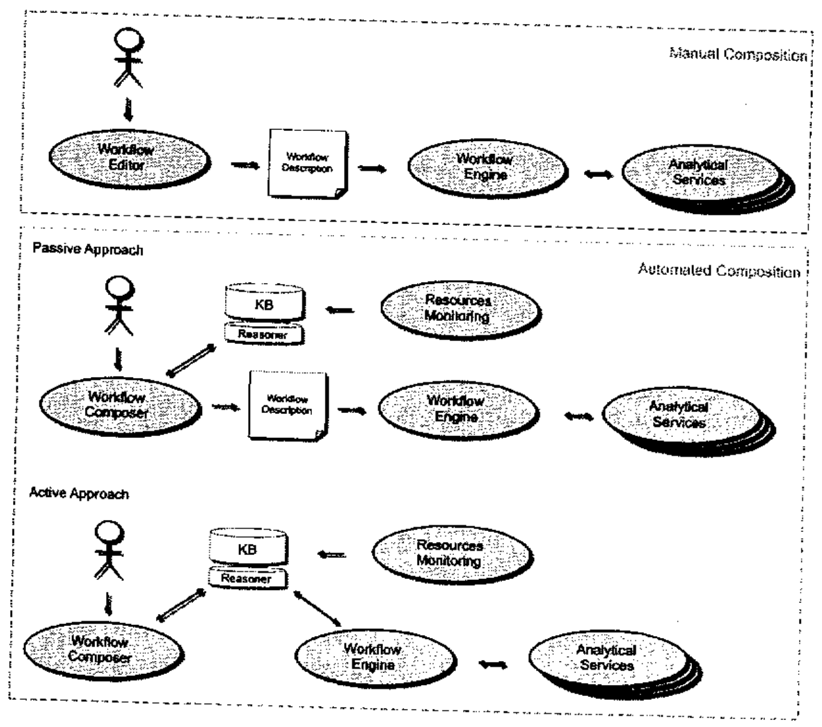
BACKGROUND

In the context of modern service-oriented Grid architectures, the data mining workflow can be seen as a collection of Grid services that are processed on distributed resources in a well-defined order to accomplish a larger and sophisticated data exploration goal. At the highest level, functions of Grid workflow management systems could be characterized into build-time functions and run-time functions. The build-time functions are concerned with defining and modeling workflow tasks and their dependencies while the run-time functions are concerned with managing the workflow execution and interactions with Grid resources for processing workflow applications. Users interact with workflow modeling tools to generate a workflow specification, which is submitted for execution to a run-time service called workflow enactment service, or *workflow engine*. Many languages, mostly based on XML, were defined for workflow description, like XLANG (Thatte, 2001), WSFL (Leymann, 2001), DSCL (Kickinger et al., 2003) and BPML (Arkin, 2002). Eventually the WSBPEL (Arkin et al., 2005) and BPEL4WS (BEA et al., 2003) specifications emerged as the de facto standard.

In our research, we consider two main workflow composition models: *manual* (implemented in the fully functional GridMiner prototype (Kickinger et al., 2003)) and *automated* (addressed in this chapter), as illustrated in Figure 1. Within manual composition, the user constructs the target workflow specification graphically in the workflow editor by means of the advanced graphical user interface. The graphical form is converted into a *workflow description* document, which is passed to the workflow engine. Based on the workflow description, the engine sequentially or in parallel calls the appropriate analytical services (database access, preprocessing, OLAP, classification, clustering, etc.). During the workflow execution, the user only has the ability to stop, inspect, resume, or cancel the execution. As a result, the user has limited abilities to interact with the workflow and influence the execution process. A similar approach was implemented in the DiscoveryNet (Sairafi et al., 2003) workflow management system.

The automated composition is based on an intensive support of five involved components: *workflow composer*, *resources monitoring*, *workflow engine*, *knowledge base*, and *reasoner*.

Figure 1. Workflow composition approaches



Workflow composer: Is a specialized tool, which interacts with a user during the workflow composition process. This chapter describes its functionality in detail.

Resources monitoring: Its main purpose is obtaining information concerning the utilization of system resources. Varieties of different systems exist for monitoring and managing distributed Grid-based resources and applications. For example, the monitoring and discovery system (MDS) is the information services component of the Globus Toolkit (Globus Alliance, 2005), which provides information about available resources on the Grid and their status. Moreover, MDS facilitates the discovery and characterization of resources and monitors services and computations. The information provided by resource monitoring can be continuously updated in the knowledge base (KB) to reflect the current status of the Grid resources.

Workflow engine: Is a runtime execution environment that performs the coordination of services as specified in the workflow description expressed in terms of a workflow language. The workflow engine is able to invoke and orchestrate the services and acts as their client, that is, listen to the notification messages, deliver outputs, and so forth.

Knowledge base (KB) and **reasoner:** A set of ontologies can be used for the specification of the KB structure, which is built-up using a set of instances of ontology classes and rules. The reasoner applies deductive reasoning about the stored knowledge in a logically consistent manner; it assures consistency of the ontology and answers given queries.

Due to different roles and behaviors of the presented components, we distinguish two modes of automated workflow composition: *passive* and *active*.

Passive Workflow Construction

The passive approach is based on the assumption that the *workflow composer* is able to compose a reasoning-based complete workflow description involving all possible scenarios of the workflow engine behavior and reflecting the status of the involved Grid resources and task parameters provided by the user at the workflow composition time. Although the KB is continuously modified by the user's entries and by information retrieved from the resource monitoring services, the composition of involved services is not updated during the workflow execution. Therefore, the composition does not reflect the 'state of the world,' which can be dynamically changed during the execution. It means that the workflow engine does not interact with the inference engine to reason about knowledge in the KB. Thus, the behavior of the engine (the decisions it takes) is steered by fixed condition statements as specified in the workflow document.

The essential tasks leading to a final outcome of the passive workflow composition approach can be summarized as follows:

1. The workflow composer constructs a complete workflow description based on the information collected in KB and presents it to the workflow engine in an appropriate workflow language.
2. The workflow engine executes each subsequent composition step as presented in the workflow description, which includes all possible scenarios of the engine behavior.

Active Workflow Construction

The active approach assumes a kind of intelligent behavior by the workflow engine supported by an inference engine and the related KB. Workflow composition is done in the same way as in the passive approach, but its usability is more efficient because it reflects a 'state of the world.' It means that the outputs and effects of the executed services are propagated to the KB together with changes of the involved Grid resources. Considering these changes, the workflow engine dynamically makes decisions about next execution steps. In this approach, no workflow document is needed because the workflow engine instructs itself using an inference engine which queries and updates the KB. The KB is queried each time the workflow engine needs information to invoke a consequent service, for example, it decides which concrete service should be executed, discovers the values of its input parameters in the KB, and so forth. The workflow engine also updates the KB when there is a new result returned from an analytical service that can be reused as input for the other services.

The essential tasks leading to a final outcome in active workflow composition approach can be summarized as follows:

1. The workflow composer constructs an abstract workflow description based on the information collected in the KB and propagates the workflow description back into the KB. The abstract workflow is not a detailed description of the particular steps in the workflow execution but instead a kind of path that leads to the demanded outcome.
2. The workflow engine executes each subsequent composition step as a result of its interaction with the KB reflecting its actual state. The workflow engine autonomously constructs directives for each service execution and adapts its behavior during the execution.

Related Work

A main focus of our work presented in this chapter is on the above mentioned passive approach of the automated workflow composition. This research was partially motivated by (Bernstein et al., 2001). They developed an intelligent discovery assistant (IDA), which provides users (data miners) with (1) systematic enumerations of valid data mining processes according to the constraints imposed by the users' inputs, the data, and/or the data mining ontology in order that important and potentially fruitful options are not overlooked, and (2) effective rankings of these valid processes by different criteria (e.g., speed and accuracy) to facilitate the choice of data mining processes to execute. The IDA performs a search of the space of processes defined by the ontology. Hence, no standard language for ontology specification and appropriate reasoning mechanisms are used in their approach. Further, they do not consider any state-of-the-art workflow management framework and language.

Substantial work has already been done on automated composition of Web services using Semantic Web technologies. For example, Majithia et al., (2004) present a framework to facilitate automated service composition in service-oriented architectures (Tsalgatidou & Pilioura, 2002) using Semantic Web technologies. The main objective of the framework is to support the discovery, selection, and composition of semantically-described heterogeneous Web services. The framework supports mechanisms to allow users to elaborate workflows of two levels of granularity: abstract and concrete workflows. Abstract workflows specify the workflow without referring to any specific service implementation. Hence, services (and data sources) are referred to by their logical names. A concrete workflow specifies the actual names and network locations of the services participating in the workflow. These two level workflow granularities are also considered in our approach, as shown in an application example.

Challenges associated with Grid workflow planning based on artificial intelligence concepts and with generation of abstract and concrete workflows are addressed by (Deelman et al., 2003). However, they do not consider any service-oriented architecture. Workflow representation and enactment are also investigated by the NextGrid

Project (NextGrid Project, 2006). They proposed the OWL-WS (OWL for workflow and services) (Beco et al., 2006) ontology definition language. The myGrid project has developed the Taverna Workbench (Oinn et al., 2004) for the composition and execution of workflows for the life sciences community. The assisted composition approach of Sirin (Sirin et al., 2004) uses the richness of Semantic Web service descriptions and information from the compositional context to filter matching services and help select appropriate services.

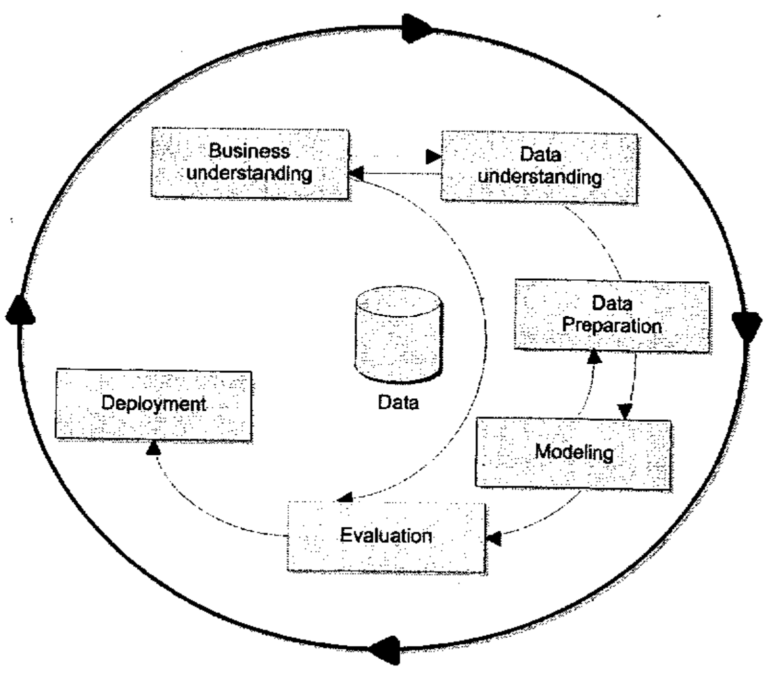
UNDERLYING STANDARDS AND TECHNOLOGIES

CRoss Industry Standard Process for Data Mining

Cross industry standard process for data mining (CRISP-DM) (Chapman et al., 1999) is a data mining process model that describes commonly used approaches that expert data miners use to tackle problems of organizing phases in data mining projects. CRISP-DM does not describe a particular data mining technique; rather it focuses on the process of a data mining projects' life cycle. The CRISP-DM data mining methodology is described in terms of a hierarchical process model consisting of sets of tasks organized at four levels of abstraction: phase, generic task, specialized task, and process instance. At the top level, the life cycle of a data mining project is organized into six phases as depicted in Figure 2.

The sequence of the phases is not strict. Moving back and forth between different phases is always required. It depends on the outcome of each phase, which one, or which particular task of a phase has to be performed next. In this chapter, we focus our attention on the three phases of data mining projects' life cycle, namely: data understanding, data preparation, and modeling.

Figure 2. Phases of CRISP-DM reference model (Chapman et al., 1999)



Data understanding: This phase starts with an initial data collection and proceeds with analytic activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

Data preparation: This phase covers all activities to construct the final data set from the initial raw data. Data preparation tasks are likely to be performed multiple times and not in any prescribed order. The tasks include table, record, and attribute selection as well as transforming and cleaning data for the modeling phase.

Modeling: In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type. Some techniques have specific requirements on the form of the data. Therefore, stepping back to the data preparation phase is often required.

The presented phases can be delimitated into a set of tasks defined by their outputs as presented in Table 1.

Predictive Model Markup Language

Predictive model markup language (PMML) (Data Mining Group, 2004) is an XML-based language that provides a way for applications to define statistical and data mining models and to share these models between PMML compliant applications. More precisely, the language's goal is to encapsulate a model in application and in a system independent fashion so that its producer and consumer can easily use it. Furthermore, the language can describe some of the operations required for cleaning and transforming input data prior to modeling. Since PMML version 3.1 is an XML based standard, its specification comes in the form of an XML schema that defines language primitives as follows:

Table 1. Generic tasks and outputs of the CRISP-DM reference model

Data Understanding	Data Preparation	Modeling
Collect Initial Data • Initial Data Collection Report **Describe Data** • Data Description Report **Explore Data** • Data Exploration Report **Verify Data Quality** • Data Quality Report	**Data Set** • Data Set Description **Select Data** • Rationale for Inclusion/ Exclusion **Clean Data** • Data Cleaning Report **Construct Data** • Derived Attributes • Generated Records **Integrate Data** • Merged Data **Format Data** • Reformatted Data	**Select Modeling Techniques** • Modeling Techniques • Modeling Assumption **Generate Text Design** • Text Design **Build Model** • Parameter Settings • Models • Model Description **Assess Model** • Model Assessment • Revised Parameter Settings

- **Data Dictionary:** It defines fields that are the inputs for models and specifies their types and value ranges. These definitions are assumed to be independent of specific data mining models. The values of a categorical field can be organized in a hierarchy as defined by the taxonomy element, and numeric fields can be specified by their intervals.
- **Mining schema:** The mining schema is a subset of fields as defined in the data dictionary. Each model contains one mining schema that lists fields as used in that model. The main purpose of the mining schema is to list fields, which a user has to provide in order to apply the model.
- **Transformations:** It contains descriptions of derived mining fields using the following transformations: normalization—mapping continuous or discrete values to numbers; discretization—mapping continuous values to discrete values; value mapping—mapping discrete values to discrete values; aggregation—summarizing or collecting groups of values, for example, compute averages; and functions—derive a value by applying a function to one or more parameters.
- **Model statistics:** It stores basic uni-variate statistics about the numerical attributes used in the model such as minimum, maximum, mean, standard deviation, median, and so forth.
- **Data mining model:** It contains specification of the actual parameters defining the statistical and data mining models. The latest PMML version addresses the following classes of models: association rules, decision trees, center-based clustering, distribution-based clustering, regression, general regression, neural networks, naive bayes, sequences, text, ruleset, and support vector machine.

The models presented in PMML can be additionally defined by a set of extensions that can increase the overall complexity of a mining model as follows:

- **Built-in functions:** PMML supports functions that can be used to perform preprocessing steps on the input data. A number of predefined built-in functions for simple arithmetic operations like sum, difference, product, division, square root, logarithm, and so forth, for numeric input fields, as well as functions for string handling such as trimming blanks or choosing substrings are provided.
- **Model composition:** Using simple models as transformations offers the possibility to combine multiple conventional models into a single new one by using individual models as building blocks. This can result in models being used in sequence, where the result of each model is the input to the next one. This approach, called 'model sequencing,' is not only useful for building more complex models but can also be applied to data preparation. Another approach, 'model selection,' is used when the result of a model can be used to select which model should be applied next.
- **Output:** It describes a set of result values that can be computed by the model. In particular, the output fields specify names, types and rules for selecting specific result features. The output section in the model specifies default names for columns in an output table that might be different from names used locally in the model. Furthermore, they describe how to compute the corresponding values.
- **Model verification:** A verification model provides a mechanism for attaching a sample data set with sample results so that a PMML consumer can verify that a model has been implemented correctly. This will make model exchange much more transparent for

users and inform them in advance in case compatibility problems arise.

Weka Toolkit

Weka (Witten & Eibe, 2005) is a collection of machine learning algorithms, especially classifications, for data mining tasks. Moreover, Weka contains tools for data preprocessing, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. The Weka's API is organized in a hierarchical structure, and the algorithms are delimitated by their relevancy to the classes of data mining tasks as presented in Figure 3.

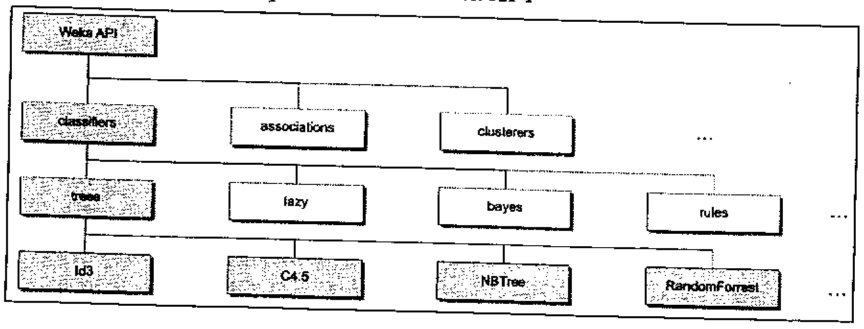
Java Data Mining Application Programming Interface

The Java data mining API (JDM) (Hornick et al., 2003) proposes a pure Java API for developing data mining applications. The idea is to have a common API for data mining that can be used by clients without users being aware or affected by the actual vendor implementations for data mining. A key JDM API benefit is that it abstracts out the physical components, tasks, and even algorithms of a data mining system into Java classes. It gives a very good basis for defining concrete data mining algorithms and describing their parameters and results. JDM does not define a large number of algorithms, but provides mechanisms to add new ones, which helps in fine tuning the existing algorithms. Various data mining functions and techniques like statistical classification and association, regression analysis, data clustering, and attribute importance are covered by this standard.

Web Ontology Language for Services

Web ontology language for services (OWL-S) **(Martin et al., 2004) consists of several interre**lated OWL ontologies that provide a set of well defined terms for use in service applications. OWL-S leverages the rich expressive power of OWL together with its well-defined semantics to provide richer descriptions of Web services that include process preconditions and effects. This enables the encoding of service side-effects that are often important for automated selection and composition of Web services. OWL-S also provides means for the description of nonfunctional service constraints that are useful for automated Web service discovery or partnership bindings. OWL-S uses OWL to define a set of classes and their properties specific to the description of Web services. The class *Service* is at the top of this ontology (see Figure 4), which provides three essential types of knowledge about a service represented as classes: *ServiceProfile*, *ServiceGrounding* and *ServiceModel*.

Figure 3. Taxonomy of algorithms as presented in Weka API



- The *ServiceProfile* describes "what the service does." The profile provides information about a service that can be used in the process of service discovery to determine whether the service meets one's needs.
- The *ServiceModel* informs "how to use the service." In more detail, the model gives information about the service itself and describes how to perform a specific task composed by subtasks involving certain conditions.
- The *ServiceGrounding* specifies the service-specific details of how to access the service, for example communication protocols, message formats, port numbers, and so forth. It is a kind of mapping from abstract activity description to its concrete implementation.

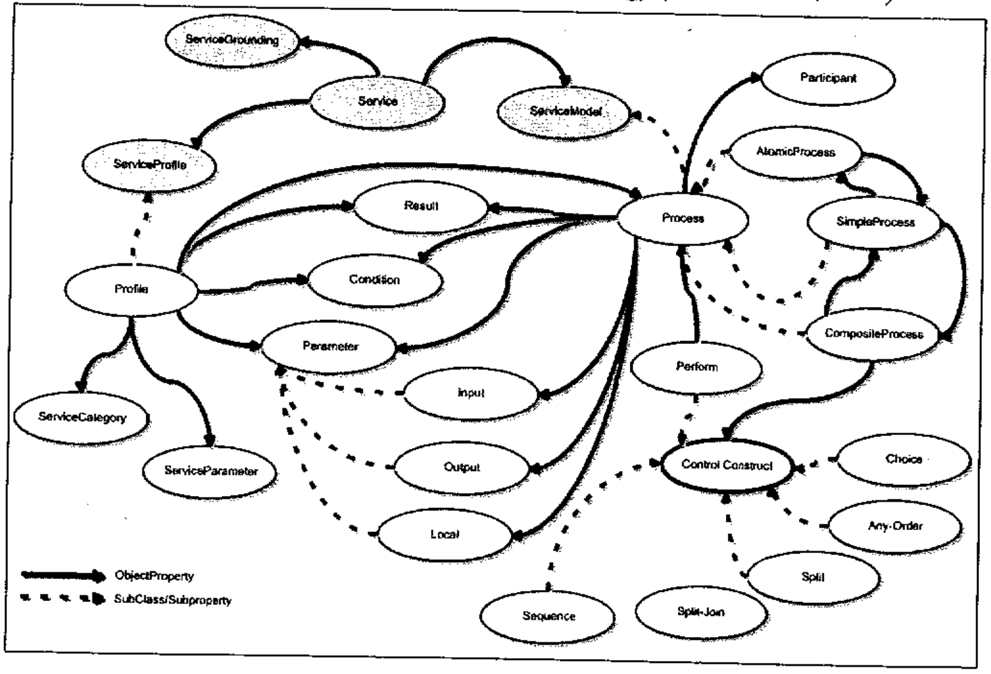
As we deal with the services composition, the aspects of *ServiceModel* and its main class *process*, including subclasses *AtomicProcess*, *SimpleProcess*, and *CompositeProcess* and their properties are discussed here in more detail.

Atomic process: The atomic process specifies an action provided by the Web service that expects one message as an input and returns one message in response. It means that the atomic processes are directly invokable and have no other subprocesses to be executed in order to produce a result. By definition, for each atomic process there must be grounding provided, which is associated with a concrete service implementation.

Simple process: The simple process gives a higher abstraction level of the activity execution. It is not associated with groundings and is not directly invokable, but like the atomic process, it is conceived of having a single step execution.

Composite process: Web services composition is a task of combining and linking Web services to create new processes in order to add value to the

Figure 4. Selected classes and their relations in OWL-S ontology (Martin et al., 2004)



collection of services. In other words, it means that composition of several services can be viewed as one composite process with its defined inputs and outputs.

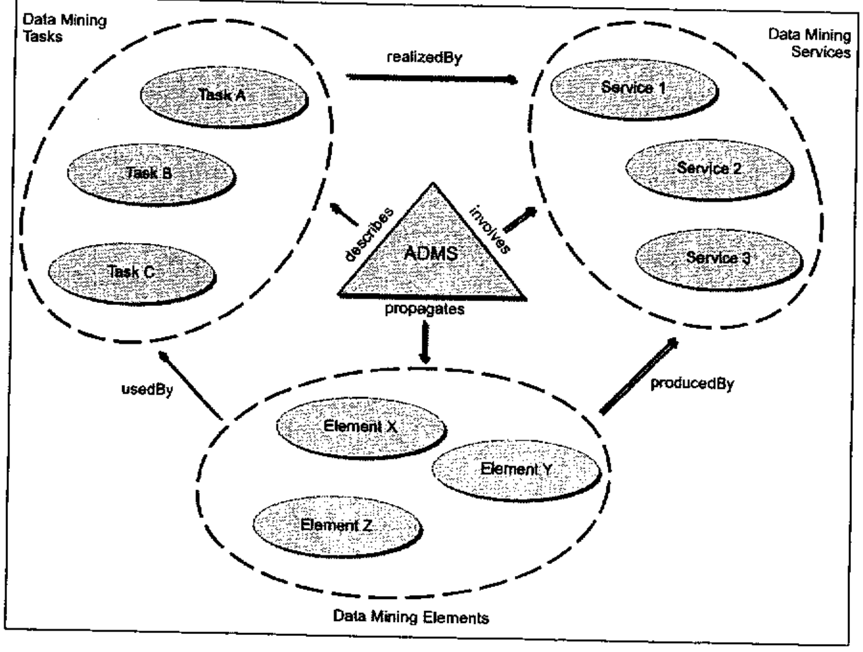
Moreover, OWL-S enables inclusion of some expressions to represent logical formulas in Semantic Web rule language (SWRL) (Horrocks et al., 2004). SWRL is a rule language that combines OWL with the rule markup language providing a rule language compatible with OWL. SWRL includes a high-level abstract syntax for Horn-like rules in OWL-DL and OWL-Lite, which are sublanguages of OWL. SWRL expressions may be used in OWL-S preconditions, process control conditions (such as if-then-else), and in effects expressions.

GRIDMINER ASSISTANT

Design Concepts

To achieve the goals presented in the Introduction section, we have designed a specialized tool—**GridMiner Assistant** (GMA)—that fulfils the role of the workflow composer shown in Figure 1. It is implemented as a Web application able to navigate a user in the phases of the knowledge discovery process (KDD) and construct a workflow consisting of a set of cooperating services aiming to realize concrete data mining objectives. The main goal of the GMA is to assist the user in the workflow composition process. The GMA provides support in choosing particular objectives of the knowledge discovery process and

Figure 5. Concept overview of the abstract data mining service



manage the entire process by which properties of data mining tasks are specified and results are presented. It can accurately select appropriate tasks and provide a detailed combination of services that can work together to create a complex workflow based on the selected outcome and its preferences. The GMA dynamically modifies the tasks composition depending on the entered values, defined process preconditions and effects, and existing description of services available in the KB. For this purpose we have designed a **data mining ontology** (DMO), which takes advantage of an explicit ontology of data mining techniques and standards (as presented in the above sections) using the OWL-S concepts to describe an abstract Semantic Web service for data mining and its main operations.

The service named **abstract data mining service** (ADMS) simplifies the architecture of the DMO as the realization of the OWL-S service with a detailed description of its *profile* and *model.* To clearly present the process of workflow composition using operations of the ADMS, we define three essential types of data mining components involved in the assisted workflow composition: DM-*elements, DM-tasks* and DM-*services,* as depicted in Figure 5. In order to design the ADMS, we consider a set of transactions representing its functionality described by DM-tasks. The DM-tasks can be seen as operations of the ADMS realized by concrete operations of involved DM-services using DM-elements as their inputs and outputs.

The following paragraphs introduce the data mining ontology, which is built through the description of the DM-tasks, DM-elements and involved DM-services. The ontology covers all phases of the knowledge discovery process and describes available data mining tasks, methods, algorithms, their inputs and results they produce. All these concepts are not strictly separated but are rather used in conjunction forming a consistent ontology.

Data Mining Elements

The DM-elements are represented by OWL classes together with variations of their representations in XML. It means that a concept described by an OWL class can have one or more related XML schemas that define its concrete representation in XML. The elements are propagated by the ADMS into the KB and can be used in any phase of data mining process. The instances of OWL classes and related XML elements are created and updated by the ADMS service operations as results of concrete services or user inputs. The elements can also determine the behavior of a workflow execution if used in SWRL rules and have an influence on preconditions or effects in the OWL-S processes. In the DMO, we distinguish two types of DM-elements: settings and results. The settings represent inputs for the DM-tasks, and on the other hand, the results represent outputs produced by these tasks. From the workflow execution point of view, there is no difference between inputs and outputs because it is obvious that an output from one process can be used, at the same time, as an input for another process. The main reason why we distinguish inputs and outputs as settings and results is to simplify the workflow composition process, to ease searching in the KB, and to exactly identify and select requested classes and their properties.

The **settings** are built through enumeration of properties of the data mining algorithms and characterization of their input parameters. Based on the concrete Java interfaces, as presented in the Weka API and JDM API, we constructed a set of OWL classes and their instances that handle input parameters of the algorithms and their default values (see Figure 6). The settings are also used to define different types of data sets that can be involved in the KDD process. Class *DataSet* and its derived subclasses collect all necessary information about the data set (file location, user name, SQL etc.) that can be represented by different data repositories such as a relational database, CSV, WebRowSet file, and so forth. Properties of the *DataSet* are usually specified by a user at the very beginning of the KDD process composition.

The following example shows a concrete instance of the OWL class *algorithm* keeping input parameters of an Apriory-type algorithm (Agrawal et al., 1994), which produces an association model. The example is presented in OWL abstract syntax (World Wide Web Consortium, 2004).

```
    Class (Setting partial Element)
    Class (Algorithm partial Element)
    Class (Parameter partial Element)

    ObjectProperty( hasParameter
domain(Setting)
     range(Parameter))

   Individual(_algorithm_AprioryType_Setting
     annotation(rdfs:label "Apriori-type algo-
rithm")
     type(Algorithm)
   value(hasParameter _number_of_rules)

   value(hasParameter _minimum_support)
   value(hasParameter _minimun_rule_confi-
dence))

   Individual(_number_of_rules
     annotation(rdfs:label "The required number
of rules")
     type(Parameter)
   value(value "10"))

   Individual(_minimum_support
     annotation(rdfs:label "The delta for minimum
support")
     type(Parameter)
   value(value "0.05"))

   Individual(_minimun_rule_confidence
     annotation(rdfs:label "The minimum confi-
dence of a rule")
     type(Parameter)
     value(value "0.9"))
```

The **results** are built on taxonomy of data mining models and characterization of their main components as presented in the PMML specification, therefore, the terminology used for naming the result elements is tightly linked with the names of the elements in PMML. As a result, it is easy to map its concepts to the concrete XML representations as done in the PMML schema. Figure 7 depicts the basic classes and their relations used to describe the Result DM-elements in the DMO.

Figure 6. Basic setting classes used to describe input parameters

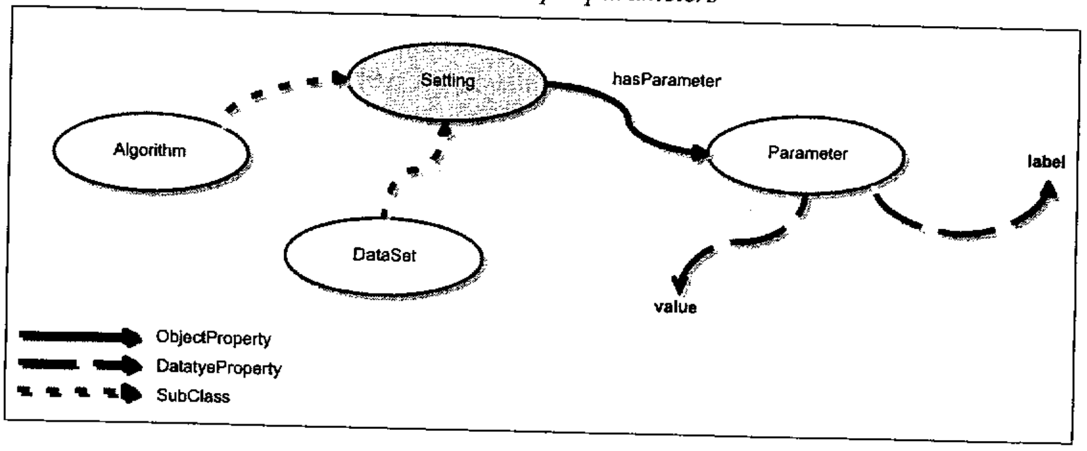
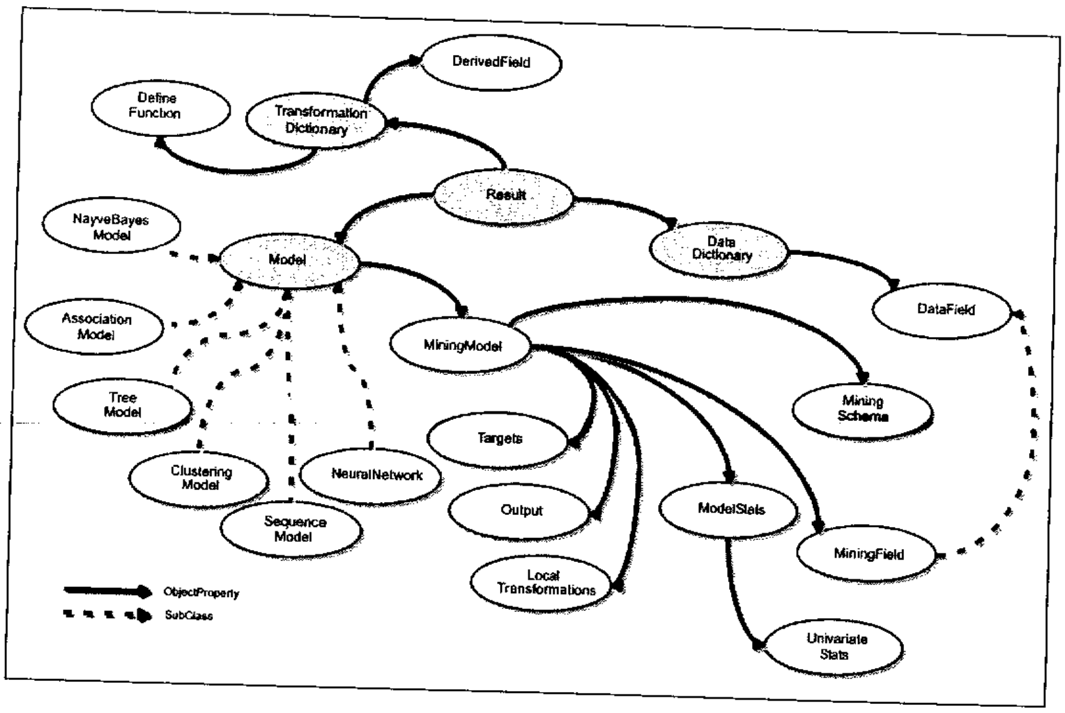


Figure 7. Basic classes used to describe Results in DMO



From the perspective of a Web service, the DM-elements can be seen as messages exchanged between service and client (XML elements), and from the abstract workflow point of view, as items exchanged between activities of simple or atomic process inside a composite process (instances of OWL classes). The following example shows how the PMML element DataDictionary, having subelements DataField and taxonomy, can be represented as *DataDictionary* class in the OWL.

DataDictionary — XML Schema:

```
<element name="DataDictionary">
   <complexType>
    <sequence>
            <element ref="DataField"
maxOccurs="unbounded" />
       <element ref="Taxonomy" minOccurs="0"
maxOccurs="unbounded" />
        </sequence>
          <attribute name="numberOfFields"
type="nonNegativeInteger" />
     </complexType>
   </element>
```

DataDictionary — OWL Class:

```
<Class rdf:ID="DataDictionary">
   <Restriction>
      <onProperty rdf:resource="#hasDataFi
eld"/>
   </Restriction>
   <Restriction>
      <onProperty rdf:resource="#hasTaxono
my"/>
      <minCardinality rdf:datatype="#nonNeg
ativeInteger">0</minCardinality>
   </Restriction>
   <Restriction>
      <onProperty rdf:resource="#numberOf
Fields"/>
   </Restriction>
  </rdfs:subClassOf>
 </Class>
```

Data Mining Tasks

The tasks are specialized operations of the ADMS organized in the phases of the KDD process as presented in the CRISP-DM reference model. The GMA composes these tasks into consistent and valid workflows to fulfill selected data mining objectives. The tasks are workflow's building blocks and are realized by concrete operations of involved DM-Services using DM-elements as their settings and results. Furthermore, GMA can automatically select and insert additional tasks into the workflow to assure validity and logical consistency of the data mining processes. We distinguish two types of DM-tasks that are forming the OWL-S *ServiceModel* of the ADMS—*setters* and *getters*.

Setters and getters give a functional description of the ADMS expressed in terms of the transformation produced by the abstract service. Furthermore, the setters are used to specify the input parameters for data mining tasks, and the getters are designed to present results of concrete service operations. The setters interact with a user who specifies values of the input parameters represented as properties of the *settings* class, for example, location of data source, selection of target attributes, the number of clusters, and so forth. The setters do not return any results but usually have an effect on creating and updating the DM-elements. The setters are not realized by concrete operations of involved services but are used to compose compact workflows and assure interaction with the user. The getters are designed to describe actual data mining tasks at different levels of abstraction. Thus a getter can be represented by an instance of the *CompositeProcess* class as, for example, a sequence of several subtasks, or a getter can be directly defined as an instance of the *AtomicProcess* class realized by a concrete operation of a DM-service.

Table 2. DM-tasks and their DM-elements

	crispdm task	gmd dm-task	input	output
data understanding	collect initial data	setdataset	datasetsettings	dataset
	describe data	getdatadictionary	dataset	datadictionary
		settaxonomy	taxonomysettings	taxonomy
	explore data	getmodelstats	dataset	modelstats
	verify data quality			
data preparation	select data	setminingschema	miningschemasettings	miningschema
	clean data	gettransformation	definefunction	dataset
	construct data		derivedfield	dataset
	integrate data		mediationschema	dataset
	format data		miningschema	dataset
modeling	select modeling technique	setminingmodel	miningmodelsettings	miningmodel
	generate test design	settestset	datasetsettings	dataset
	build model	getclassificationmodel getassociationmodel getclusteringmodel getsequentialmodel getneuralnetworksmodel	mininingmodelsettings	model
	assess model	getmodelverification	mininingmodelsettings	model

Table 2 presents some of the setters and getters on the highest level of abstraction organized according to the phases of the CRISP-DM reference model and lists their input and output DM-elements.

The setters are designed to interact with the user, therefore, each *setter* has a related HTML input form used by the user to insert or select the input parameters' values of the examined DM-element. The GMA presents the form implemented as a dynamic Web page to the user, and based on his/her inputs, the GMA updates parameters of the DM-elements.

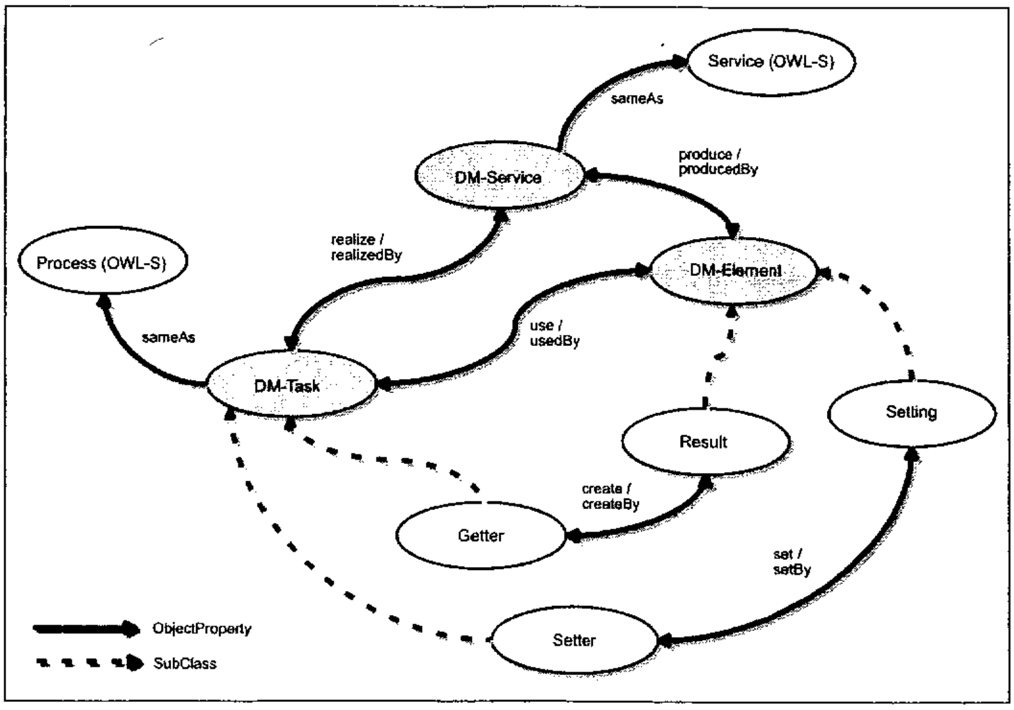
Data Mining Services

Realization of a particular DM-task is done by invoking concrete operations of involved DM-services described in OWL-S as an atomic, simple or composite process related to its *ServiceGrounding* (operators that can be executed) as defined in the appropriate WSDL document. The operations produce DM-elements that can be reused by other operations in further steps of the workflow execution. Within our project, several data mining services were developed including decision tree, clustering, associations, sequences, and neural networks with detailed descriptions of their functionality in OWL-S.

Data Mining Ontology

Based on the concepts and principal classes in the preceding sections, we have constructed the final DMO as depicted in Figure 8. The DMO incorporates the presented OWL-S ontology and its classes describing DM-tasks and DM-services as well as *Result* and *Setting* classes, which describe the DM-elements. The ontology is also supplemented by a set of semantic rules that determine in detail particular relations between involved classes, but its presentation is out of the scope of this chapter.

Figure 8. Basic classes and their relations in DMO



WORKFLOW CONSTRUCTION

In order to create the final workflow, the GMA follows a combination of the backward and forward chaining approaches. It means that the process begins with a user-based selection of a target task, which produces the desired data exploration output. Additional tasks are automatically inserted into a chain before the target task until a task without any or already satisfied preconditions is encountered (backward phase). Next, by insertion of additional tasks, this chain is automatically extended into a form in which all matching preconditions and inputs parameters are satisfied (forward chain). According to this, our approach to workflow construction is based on two phases as follows.

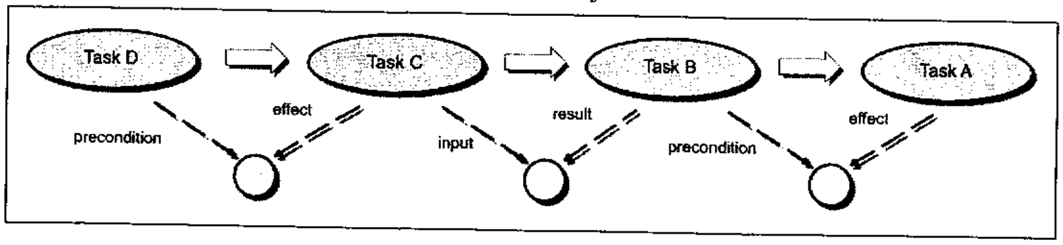
Tasks Composition

The aim of this phase is to create an abstract workflow consisting of a sequence of DM-tasks. Figure 9 presents an example of the abstract workflow composed of DM-tasks. 'Task D' is the initial task inserted into the workflow in the sense of the previously mentioned backward phase of the workflow composition, and the task's result, represented by a DM-element, is the final goal of the abstract workflow. The DM-element can be, for example, a decision tree model in the data mining phase, a list of all available statistics in the data understanding phase, or the data preparation phase can result in a new transformed data set. Selection of the final result is the only interaction with the user in this phase; the other steps are hidden. The composition then continues with an examination of preconditions and inputs of the target task 'Task D.' If the task has an input which does not exist (KB does not contain an instance of the required DM-element) or condition that has to be satisfied, then the KB is queried for such a task that can supply the required DM-elements or can satisfy these preconditions by its effects; the missing task can be 'Task C' in our case. The design of the ontology ensures that there is only one such task that can be selected and inserted into the workflow prior to the examined task. For example, if we want to obtain a list of statistics (getModelStats task) then there must be an existing DM-element DataSet. It means that a task which creates the DataSet element must anticipate the getModelStats task in the workflow composition (it can be the setDataSet task in our case). The newly added tasks are treated in the same way until a task without any preconditions or already satisfied preconditions is encountered, or a task without any input that is produced as result of another task is reached, which is 'Task A' in our example.

Values Acquisition

Figure 10 presents the same workflow but viewed from another perspective: now 'Task A' is the initial task and 'Task D' is the final one. In this phase of the workflow construction, the task parameters are set up. Their values can be obtained in the following ways: (a) as effects of DM-tasks (getters) or (b) entered directly by a user (setters).

Figure 9. Example of tasks composing the abstract workflow



In other words, not all values of input parameters can be obtained automatically as results of previous operations and therefore must be supplied by a user. This phase of the values acquisition starts by tracing the abstract workflow from its beginning, 'Task A', and supplying the values by abstract interpretation of the partial workflow or providing them from a user. The user can enter the values directly by filling input fields offered by an appropriate graphical user interface or by selecting them from a list created as a result of a KB query, e.g., a list of data mining algorithms for a specific method is determined by available implementations of services able to perform the task. If the user selects a list item value that has influence on the precondition or effect that has to be satisfied in the next steps, then the KB is searched for such a task that can satisfy this request. The newly discovered tasks are inserted automatically into the workflow. It can be, for example, a case when the user wants to increase the quality of used data adding some transformation tasks, presenting the resulting model in different form, and so forth.

To illustrate the main features of the GMA and explain the phases of the tasks composition and values acquisition, we present a practical scenario addressing step-by-step construction of a simple workflow aiming at discovering of classification model for a given data set. This scenario is taken from a medical application dealing with patients suffering from serious traumatic brain injuries (TBI).

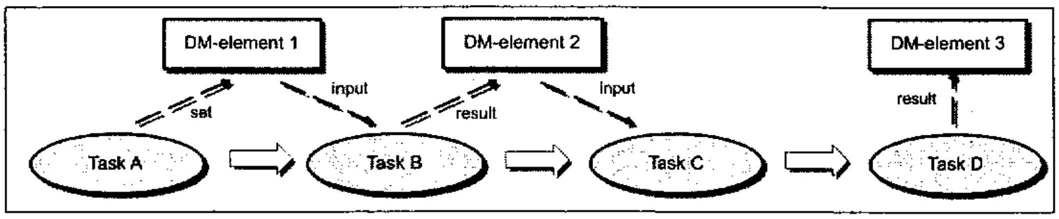
Workflow Construction Example

At the first clinical examination of a TBI patient (Brezany et al., 2003), it is very common to assign the patient into a category, which allows to define his/her next treatment and helps to predict the final outcome of the treatment. There are five categories of the final outcome defined by the Glasgow outcome scale (GOS): dead, vegetative, severely disabled, moderately disabled, and good recovery.

It is obvious that the outcome is influenced by several factors that are usually known and are often monitored and stored in a hospital data warehouse. For TBI patients, these factors are for example: injury severity score (ISS), abbreviated injury scale (AIS), Glasgow coma score (GCS), age, and so forth. It is evident that if we want to categorize the patient, then there must be a prior knowledge based on cases of other patients with the same type of injury. This knowledge can be mined from the historical data and represented as a classification model. The mined model is then used to assign the patient to the one of the outcome categories. In particular, the model can assign one of the values from the GOS to a concrete patient.

As we mentioned in the previous section, in the first phase, the composition of the abstract workflow proceeds by using the backward chaining approach starting with the task and then producing the demanded result. In our case, the classification model is represented by a decision

Figure 10. Example of values acquisition phase



tree. Moreover, in this example, we assume that the data understanding phase of the KDD process was successfully finished, and we have all the necessary information about the data set to be mined. It means that appropriate records corresponding to the *DataSet* and *DataDictionary* DM-elements are already available in the KB, and the workflow can start with the data preprocessing task.

Phase 1: Tasks Composition

As we presented previously, the first step of the task composition is the interactive selection of the final model from a list of all available models. The list can be obtained as a result of the following SPARQL (SPARQL, 2006) query returning a list of DM-tasks and models they produce. This query is issued by the GMA automatically. (See Box 1.)

Selection of the classification model gives us a direct link to the *getClassificationModel* DM-task that can be realized by a concrete service operation. Information about its input DM-elements and the corresponding DM-task producing them can be retrieved from the KB by submitting the following SPARQL query, which is also issued by the GMA automatically (see Box 2).

Box 1.

Query:

```
PREFIX dmo: <http://dmo.gridminer.org/v1#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?ModelName ?Task
FROM     <http://www.gridminer.org/dmo/v1/dmo.owl>
WHERE {
            ?model rdf:type <#Model> .
            ?model rdfs:label ?ModelName .
            ?model dmo:createdBy ?Task
      }
ORDER BY ?ModelName
```

Result:

ModelName T	**ask**
Association Model	getAssociationModel
Classification Model	getClassificationModel
Clustering Model	getClusteringModel
…	…

Box 2.

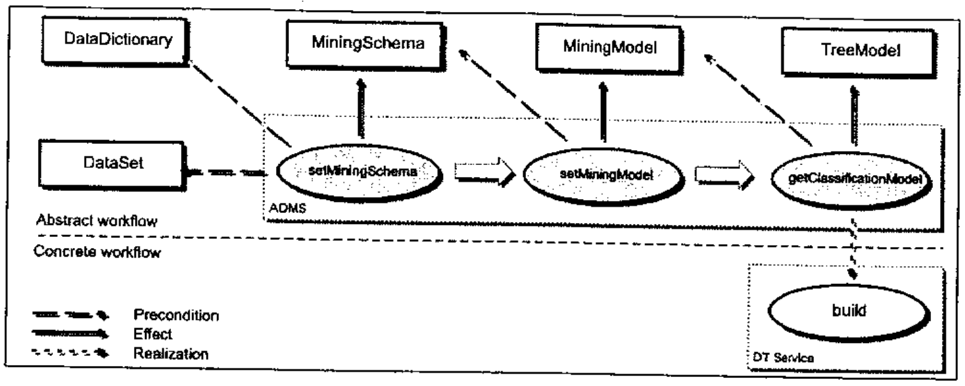
Query:

```
PREFIX dmo: <http://dmo.gridminer.org/v1#>
SELECT ?Setting ?Task
FROM     <http://www.gridminer.org/dmo/v1/dmo.owl>
WHERE {
            dmo:getClassificationModel dmo:hasSettings ?Setting .
            ?Task dmo:create ?Setting
      }
```

Result:

Setting	**Task**
MiningModel	setMiningModel

Figure 11. Abstract workflow after the phase of tasks composition



The discovered DM-task *setMiningModel* is inserted into the workflow prior to the *getClassificationModel* task, and its preconditions and inputs are examined. The only precondition of the *setMiningModel* task is the existence of the *MiningSchema* DM-element. This requirement can be satisfied by inserting the *setMiningSchema* task into the workflow, whose effect is the creation of the *MiningSchema* DM-element. The *setMiningSchema* task has two preconditions: the existence of the *DataSet* and *DataDictionary* DM-elements. Their corresponding records are already available in the KB, so no additional tasks are inserted into the abstract workflow. As the result, an abstract workflow consisting of three DM-tasks (see Figure 11) is created and is instanced as a new composite process of the ADMS in the KB. The figure also presents the DM-elements identified during the composition phase as preconditions of the involved tasks and a fragment of the concrete workflow.

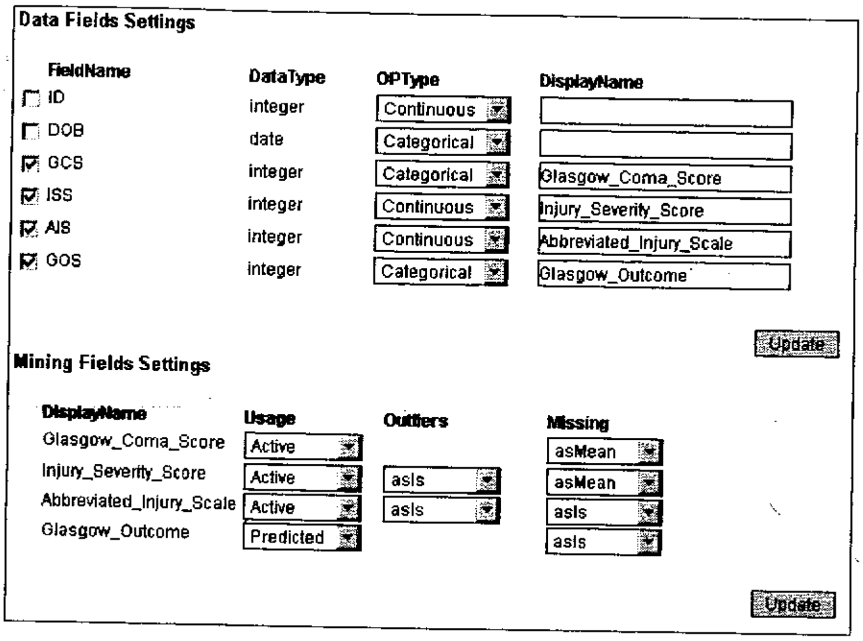
Phase 2: Values Acquisition

The second phase of the workflow construction starts with the examination of the first DM-task in the abstract workflow (*setMiningSchema*). In this phase, the values of the DM-elements' properties, identified in the previous phase, are supplied by the user and additional DM-tasks are inserted as needed. The following paragraphs describe in more detail the steps of setting the DM-elements produced and used by the involved tasks.

setMiningSchema: This task can be seen as a simple data preprocessing step where data fields (attributes) used in the modeling phase can be selected and their usage types can be specified. The primary effect of this task is a new *MiningSchema* element instanced in the KB, keeping all the schema's parameters specified by the user. Moreover, the user can specify whether some preprocessing methods should be used to treat missing values and outliers of the numerical attributes. Selection of a preprocessing method requires an additional DM-task, which is able to perform the data transformations and produce a new data set that can be used in the next steps. If one of the transformation methods is selected then the KB is queried again for a task able to transform the data set. The *getTransformation* task has the ability to transform the selected data set, therefore, can be inserted into the abstract workflow in the next step.

As we presented in previous paragraphs, the *setters* are designed to interact with the user, therefore, each *setter* has a related HTML input form used by the user to insert or select the values of the examined DM-element input parameters. The GMA presents the form implemented as a

Figure 12. Input HTML form for the MiningSchema



Data Fields Settings

FieldName	DataType	OPType	DisplayName
☐ ID	integer	Continuous	
☐ DOB	date	Categorical	
☑ GCS	integer	Categorical	Glasgow_Coma_Score
☑ ISS	integer	Continuous	Injury_Severity_Score
☑ AIS	integer	Continuous	Abbreviated_Injury_Scale
☑ GOS	integer	Categorical	Glasgow_Outcome

Mining Fields Settings

DisplayName	Usage	Outliers	Missing
Glasgow_Coma_Score	Active		asMean
Injury_Severity_Score	Active	asIs	asMean
Abbreviated_Injury_Scale	Active	asIs	asIs
Glasgow_Outcome	Predicted		asIs

dynamic Web page to the user, and based on its inputs, the GMA updates values of the DM-elements' parameters.

Figure 12 presents the input form used by the GMA to construct the *MiningSchema* DM-element. In this form, there is one mandatory property for the classification task — 'target attribute.' It is one of the categorical *DataFields* from the *DataDictionary* element, which is the GOS in our case. Therefore, the 'target attribute' must be marked as 'predicted' in the *MiningSchema* DM-element. The effect of the *setMiningSchema* task is a newly created DM-element *MiningSchema,* which describes mined fields and their transformations.

getTransformation: This task is inserted into the workflow right after the *setMiningSchema* task. It does not require interaction with the user because its input parameters are already specified in the *MiningSchema* created as the effect of the previous task. The task just examines the *MiningSchema* element and selects a concrete operation from DM-Services available in the KB, which can satisfy the chosen data preprocessing objectives. The task can select operation 'transform' of the specialized *DataPreprocessing* service (DPP service) and insert it into the concrete workflow (see Figure 14).

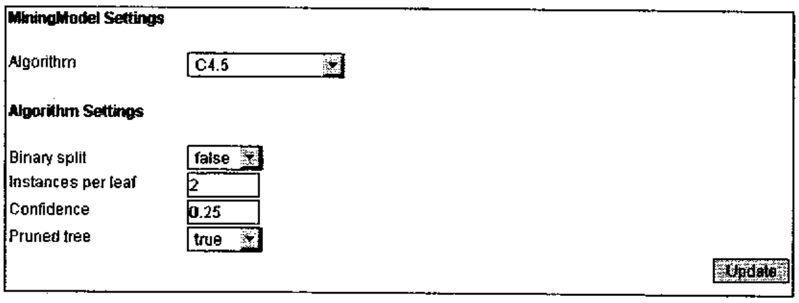
setMiningModel: Specification of the properties of the selected model is the main purpose of this task. The GMA presents a list of all available data mining algorithms producing classification models and selects its input parameters. Based on the selected parameters, a new DM-element *MiningModel* describing model properties is created as an effect of this task. The following SPARQL query retrieves all parameters for the C4.5 classification algorithm (Quinlan, 1993) that is used to setup the *MiningModel* element in our example. (See Box 3.)

The GMA presents the results to the user in the HTML form presented in Figure 13, where the user specifies values of the input parameters

Box 3.

```
Query:

        PREFIX dmo: <http://dmo.gridminer.org/v1#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        SELECT ?ParameterName ?DefaultValue
        FROM      <http://www.gridminer.org/dmo/v1/dmo.owl>
        WHERE {
                    dmo:_algorithm_c4.5_Settings dmo:hasParameter ?Parameter .
                    ?Parameter rdfs:label ?ParameterName .
                    ?Parameter dmo:value ?DefaultValue
              }
        ORDER BY ?ParameterName
```

Figure 13. Input HTML form for the MiningModel



MiningModel Settings

Algorithm	C4.5

Algorithm Settings

Binary split	false
Instances per leaf	2
Confidence	0.25
Pruned tree	true

needed to build the classification model using the C4.5 algorithm.

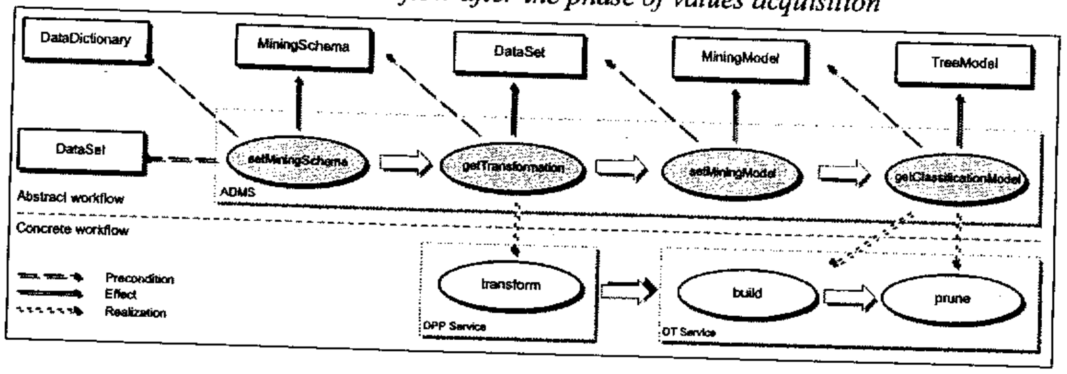
getClassificationModel: This task examines the *MiningModel* element created in the previous task and identifies the appropriate operation that can build the classification model using *MiningModel* parameters. The task can be the operation 'build' implemented by the DecisionTree Service (DT Service), which returns the classification model represented by the PMML element TreeModel. Moreover, if parameter 'pruned tree' is marked as true (false by default) then the additional operation of the DT Service 'prune' is inserted into tò the concrete workflow to assure that the discovered decision tree is modified using a pruning mechanism.

If all required parameters and preconditions of the tasks involved in the abstract workflow are satisfied then the GMA constructs a concrete workflow specification in the BPEL language and presents it to the workflow engine. The concrete workflow is a sequence of the real services and is related to the abstract DM-tasks as presented in Figure 14.

The final output returned from the workflow engine is a PMML document containing a TreeModel element that represents the demanded model that can be used to classify a particular patient into the GOS category.

The following BPEL document created in our scenario contains five variables representing the DM-elements used as inputs and outputs of the invoked operations. The variable DataSet is an

Figure 14. Abstract and concrete workflow after the phase of values acquisition



XML in WebRowSet format (`RowSet` Java object in XML format) storing all the initial data. TransformedDataset is a new WebRowSet created by the 'transform' operation, and TreeSettings is used as input for the 'build' and 'prune' operations. The variable TreeModel stores the PMML document with the full decision tree, and the PrunedTreeModel stores its pruned version. The BPEL flow reflects the composition as done in the concrete workflow consisting of three operations invoked in sequence. (See Box 4.)

SYSTEM PROTOTYPE

An overview of the first system prototype is shown in Figure 15. We use the OWL editor Protégé (Noy et al., 2001) to create and maintain the DMO, which is stored in the KB. To reason about knowledge in the KB, we use the Pellet reasoner (Sirin & Parsia, 2004), which is an open-source Java based OWL DL reasoner and provides a description logic interface (DIG) (Bechhofer et al., 2003). The GMA is implemented as a standalone Web application supported by the Jena Toolkit (McBride, 2003) and is able to interact with a user to assemble the required information. The GMA communicates over the DIG interface with the reasoner, which is able to answer a subset of RDQL queries (Seaborn, 2004). The GMA queries KB every time it needs to enumerate some parameters or find a data mining task, algorithm, service, and so forth. Moreover, the GMA also updates the KB with instances of DMO classes and values of their properties. The final outcome of the GMA is a workflow document presented to the workflow engine Auriga (Brezany et al., 2006) in the BPEL4WS language. The GMA also acts as a client of the workflow engine, which executes appropriate services as described in the BPEL document and returns their outputs back to the GMA. A more detailed characterization of these major components follows.

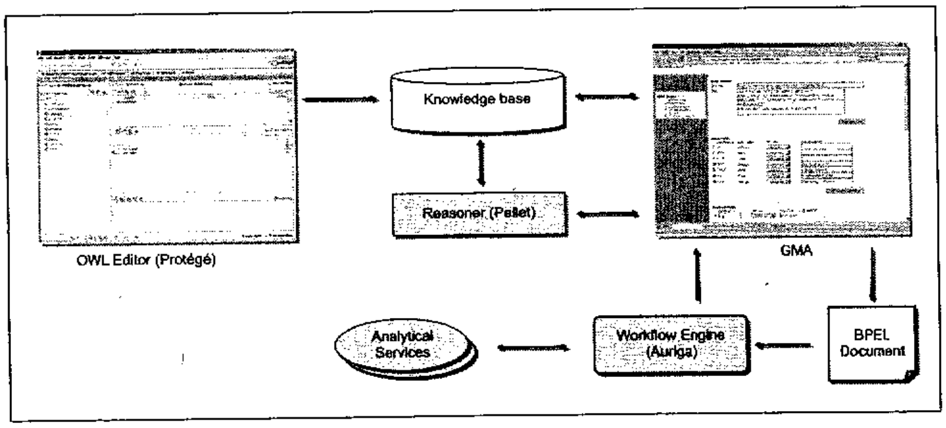
Auriga WEEP workflow engine is an easy to execute and manage workflow enactment service for Grid and Web services. The core of the engine is implemented as a standalone application referred to as the Auriga WEEP Core, which orchestrates the services as specified in a BPEL. Auriga WEEP has also a specialized version, which is wrapped by a Grid service implementation focused on using the Globus 4 container as the running environment. The engine has a pluggable architecture, which allows additional Grid specific functionality to be used in the Auriga Core extensions.

- **Jena** is a Java framework for building Semantic Web applications. It provides a programming environment for RDF, RDFS, OWL and SPARQL and includes a rule-based inference engine.

Box 4.

```
Variables:

<variable name="DataSet" element="wrs:webRowSet"/>
<variable name="TransformedDataset" element="wrs:webRowSet "/>
<variable name="TreeModel" element="pmml:TreeModel"/>
<variable name="PrunedTreeModel" element="pmml:TreeModel"/>
<variable name="TreeSettings" element="dmo:Setting"/>

Sequence:

<sequence>
    < flow>
       <invoke partnerLink="DPPService" operation="transform"   inputVariable="DataSet"
       outputVariable="TransformedDataset" />

       <invoke partnerLink="DTService" operation="build"
       inputVariable="TreeSettings" outputVariable="TreeModel"/>

       <invoke partnerLink="DTService" operation="prune"
       inputVariable="TreeSettings"  outputVariable="PrunedTreeModel"/>
    </ flow>
</ sequence>
```

Figure 15. Overview of the prototype system



- **Pellet** provides functionalities to see the species validation, check consistency of ontologies, classify the taxonomy, check entailments and answer a subset of RDQL queries. Pellet is based on the tableaux algorithms developed for expressive Description Logics and supports the full expressivity of OWL DL.
- **Protégé** is an ontology editor and knowledge acquisition system. It implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats including OWL.

FUTURE WORK

We envision the following key directions for future extension of the research presented in this chapter:

- **Active workflow engine:** This approach was already briefly discussed in the background section and sketched in Figure 1. In this case, the interaction mode between the user, workflow composer and the functionality of the composer basically remain the same as in the described passive approach. The functionality of the existing GridMiner workflow engine will be extended to be able to make dynamic decisions about the next workflow execution step based on the actual context of the knowledge base and the results of the reasoning. Moreover, the workflow composer can listen to the changes in the knowledge base and automatically interact with the user when some additional information or hints have to be supplied.
- **Workflow ranking:** The data mining ontology will be extended by estimations of each operation's effects on workflow attributes such as speed, model accuracy, etc. Due to the user's preferences (e.g., speed vs. accuracy) the composer can then better optimize individual selection steps, derive a set of workflows with the corresponding ranking and supply the best option to the workflow engine. In this process, information about the current Grid resource utilization provided by standard Grid information services can also be included into this optimization process.
- **Workflow planning:** We consider upgrading the intelligence of the workflow composer with the development of a supporting planning system which will be able to propose an abstract workflow from the specification of the goals and the initial state. We will exploit and adapt AI planning optimizations.
- **Support by autonomic computing:** We will investigate how the presented framework should be extended to be able to include some functionality of autonomic computing into the workflows composed. This involves investigating workflow patterns, categorizing requirements and objectives, and designing corresponding rule templates.

CONCLUSION

The characteristics of data exploration in scientific environments impose unique requirements for workflow composition and execution systems. In this chapter, we addressed the issues of composing workflows with automated support developed on top of Semantic Web technologies and the workflow management framework elaborated in our Grid data mining project. The kernel part of that support is a tool called the GridMiner workflow assistant (GMA), which helps the user interactively construct workflow description expressed in a standard workflow specification language. The specification is then passed to the workflow engine for execution. The GMA operations are controlled by the data mining ontology based on the concepts of PMML, JDM, WEKA and CRISP-DM. A practical example taken from a medical application addressing management of patients with traumatic brain injuries illustrates the use of the GMA. The results achieved will be extended in our future research whose key issues were outlined in the chapter. Although this research is conducted in the context of the GridMiner project, its results can be used in any system involving workflow construction activities.

FUTURE RESEARCH DIRECTIONS

In this section, we identify three future challenges and research problems in the ontology-based workflow construction and execution.

1. Extended Data Mining Ontology

Data mining as a scientific discipline is a huge domain which is still expanding. New approaches to data analyses, visualization techniques, or even new algorithms are continuously being developed. There are also plenty of real applications tailored to the application domain specifically for data mining tasks. Therefore, it is nearly impossible to completely describe this dynamic field of data mining with a static ontology. The ontology proposed in our chapter can only be used for a subset of the high number of data mining tasks. Hence we see new opportunities in extending the proposed data mining ontology with different, application domain specific, tasks that would better express the functionality of the constructed workflows.

2. Quality of Services and Workflows

Another issue that is not fully covered in the proposed ontology is the description of the quality of the involved data mining services. Especially in the Grid infrastructures, the properties of the involved resources (e.g., performance, price, bandwidth, etc.) play the crucial role in their discovery and right selection. So we see another opportunity in the detailed description of the data mining services' properties which can be done as a direct extension of the OLW-S language. Moreover, there can also be a detailed description of the composed workflows' quality which can be used for effective ranging of the entire workflows.

3. Autonomic Behavior of the Workflow Enactment Engine

Autonomic computing is one of the hottest topics in information technologies. Different areas in computer science, ranging from hardware to software implementation on the application level, try to apply some autonomic features (like, e.g., self-tuning, self-configuration, self-healing, etc.) to assure stability and availability of the system. The autonomic behavior of the Workflow Engine can ensure that the execution of the data mining workflows results in a required goal even in such a dynamic environment as the Grid where the Workflow Engine must react to the changes of the involved resources and adopt its behavior to new conditions and reflect the actual 'State of the Grid'.

REFERENCES

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Databases* (pp. 478-499). Santiage, Chile: Morgan Kaufmann.

Antonioletti, M., Krause, A., Paton, N. W., Eisenberg, A., Laws, S., Malaika, S., et al. (2006). The WS-DAI family of specifications for web service data access and integration. *ACM SIGMOD Record, 35*(1), 48-55.

Arkin, A. (2002). *Business process modeling language* (BPML). Specification. BPMI.org.

Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., et al. (2005). *Web services business process execution language version* 2.0. wsbpel-specificationdraft-01, OASIS.

BEA, IBM, Microsoft, SAP, & Siebel. (2003). *Business process execution language for Web services.* Version 1.1. Specification. Retrieved May 15, 2006, from ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

Bechhofer, S., Moller, R., & Crowther, P. (2003). *The DIG description logic interface.* International Workshop on Description Logics, Rome, Italy.

Beco, S., Cantalupo, B., Matskanis, N., & Surridge M. (2006). *Putting semantics in Grid workflow management: The OWL-WS approach.* GGF16 Semantic Grid Workshop, Athens, Greece.

Bernstein, A., Hill, S., & Provost, F. (2001). An intelligent assistant for the knowledge discovery process. *In Proceedings of the IJCAI-01 Workshop on Wrappers for Performance Enhancement in KDD*. Seattle, WA: Morgan Kaufmann.

Brezany, P., Tjoa, A.M., Rusnak, M., & Janciak, I. (2003). Knowledge Grid support for treatment of traumatic brain injury victims. *International Conference on Computational Science and its Applications*. Montreal, Canada.

Brezany, P., Janciak, I., Woehrer, A., & Tjoa, A.M. (2004). *GridMiner: A Framework for knowledge discovery on the Grid - from a vision to design and implementation*. Cracow Grid Workshop, Cracow, Poland: Springer.

Brezany, P., Janciak, I., Kloner, C., & Petz, G. (2006). *Auriga — workflow engine for WS-I/WS-RF services*. Retrieved September 15, 2006, from http://www.Gridminer.org/auriga/

Bussler ,C., Davies, J., Dieter, F., & Studer , R. (2004). The Semantic Web: Research and applications. In *Proceedings of the 1st European Semantic Web Symposium, ESWS. Lecture Notes in Computer Science, 3053*. Springer.

Chapman, P., Clinton, J., Khabaza, T., Reinartz, T., & Wirth. R. (1999). *The CRISP-DM process model*. Technical report, CRISM-DM consortium. Retrieved May 15, 2006, from http://www.crisp-dm.org/CRISPWP-0800.pdf

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web Services Description Language* (WSDL) 1.1. Retrieved May 10, 2006, from http://www.w3.org/TR/wsdl

Data Mining Group. (2004). *Predictive model markup language*. Retrieved May 10, 2006, from http://www.dmg.org/

Deelman, E., Blythe, J., Gil, Y., & Kesselman, C. (2003). Workflow management in GriPhyN. *The Grid Resource Management*. The Netherlands: Kluwer.

Globus Alliance (2005). *Globus Toolkit 4*. http://www.globus.org

Globus Alliance, IBM, & HP (2004). *The WS-Resource framework*. Retrieved May 10, 2006, from http://www.globus.org/wsrf/

Hornick, F. M., et al. (2005). *Java data mining 2.0*. Retrieved June 20, 2006, from http://jcp.org/aboutJava/communityprocess/edr/jsr247/

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A Semantic Web rule language combining OWL and RuleML*. W3C Member Submission. Retrieved May 10, 2006, from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521

Kickinger, G., Hofer, J., Tjoa, A.M., & Brezany, P. (2003). Workflow mManagement in GridMiner. *The 3rd Cracow Grid Workshop*. Cracow, Poland: Springer.

Leymann, F. (2001). *Web services flow language (WSFL 1.0)*. Retrieved September 23, 2002, from www4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Majithia, S., Walker, D. W., & Gray, W.A. (2004). *A framework for automated service composition in service-oriented architectures* (pp. 269-283). ESWS.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., et al.(2004). Bringing semantics to Web services: The OWL-S approach. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*. San Diego, California.

McBride, B. (2002). Jena: A Semantic Web toolkit. *IEEE Internet Computing*, November /December, 55-59.

Oinn, T. M., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, R. M., et al. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics, 20*(17), 3045-3054.

Noy, N. F. , Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., & Musen, M.A. (2001). Creating Semantic Web contents with Protege-2000. *IEEE Intelligent Systems, 16*(2), 60-71.

Quinlan, R. (1993). *C4.5: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann Publishers.

Sairafi, S., A., Emmanouil, F. S., Ghanem, M., Giannadakis, N., Guo, Y., Kalaitzopolous, D., et al. (2003). The design of discovery net: Towards open Grid services for knowledge discovery. *International Journal of High Performance Computing Applications, 17*(3).

Seaborne, A. (2004). *RDQL: A query language for RDF.* Retrieved May 10, 2006, from http://www.w3.org/Submission/RDQL/

Sirin, E., & Parsia, B. (2004). *Pellet: An OWL DL Reasoner, 3rd International Semantic Web Conference*, Hiroshima, Japan. Springer.

Sirin, E.B. Parsia, B., & Hendler, J. (2004). Filtering and selecting Semantic Web services with interactive composition techniques. *IEEE Intelligent Systems, 19*(4), 42-49.

SPARQL: Query Language for RDF, W3C Working Draft 4 October 2006. Retrieved October 8, 2006, from http://128.30.52.31/TR/rdf-sparql-query/

Thatte, S. (2001). *XLANG: Web services for business process design.* Microsoft Corporation, Initial Public Draft.

Tsalgatidou, A., & Pilioura, T. (2002). An overview of standards and related technology in web services. *Distributed and Parallel Databases. 12*(3).

Witten, I.H., & Eibe, F. (2005). *Data mining: Practical machine learning tools and techniques.* (2nd ed.). San Francisco: Morgan Kaufmann.

World Wide Web Consortium. (2004). *OWL Web ontology language semantics and abstract syntax.* W3C Recommendation 10 Feb, 2004.

ADDITIONAL READING

For more information on the topics covered in this chapter, see http://www.Gridminer.org and also the following references:

Alesso, P. H., & Smith, F. C. (2005). *Developing Semantic Web services.* A.K. Peterson Ltd.

Antoniou, G., & Harmelen, F. (2004). *A Semantic Web primer.* MIT Press.

Davies, J., Studer, R., & Warren P. (2006). *Semantic Web technologies: Trends and research in ontology-based systems.* John Wiley & Sons.

Davies, N. J., Fensel, D., & Harmelen, F. (2003). *Towards the Semantic Web: Ontology-driven knowledge management.* John Wiley & Sons.

Foster, I., & Kesselman, C. (1999). *The Grid: Blueprint for a new computing infrastructure.* Morgan Kaufmann.

Fox, G.C., Berman, F., & Hey, A.J.G. (2003). *Grid computing: Making the global infrastructure a reality.* John Wiley & Sons.

Han, J., & Kamber, M. (2000) *Data mining: Concepts and techniques.* Morgan Kaufmann.

Lacy, L.W. (2005). *Owl: Representing information using the Web ontology language.* Trafford Publishing.

Li, M., & Baker, M. (2005). *The Grid: Core technologies.* John Wiley & Sons.

Marinescu, D.C. (2002) *Internet-based workflow management: Toward a Semantic Web.* John Wiley & Sons.

Matjaz, B.J., Sarang, P.G., & Mathew, B. (2006). *Business process execution language for Web services* (2nd ed.). Packt Publishing.

Murch, R. (2004). *Autonomic computing.* Published by IBM Press.

Oberle, D. (2005). *The semantic management of middleware.* Springer.

Singh, M.P., & Huhns, M.N. (2006). *Service-oriented computing: Semantics, processes, agents.* John Wiley & Sons.

Sotomayor, B., & Childers, L. (2006). *Globus Toolkit 4: Programming Java services.* Morgan Kaufmann.

Stojanovic, Z., & Dahanayake. A. (2005). *Service oriented software system engineering: Challenges and practices.* Idea Group Inc.

Taylor, I. J., Deelman E., Gannon, D. B., & Shields, M. (2007). *Workflows for e-science.* Springer.

Zhong, N., Liu, J., & Yao, Y.(2003). *Web intelligence.* Springer.

Zhu, X., & Davidson, I. (2007). *Knowledge discovery and data mining: Challenges and realities.* Idea Group Inc.

Zhuge, H. (2004). *The knowledge Grid.* World Scientific.