A Language-Based and Process-Oriented Approach for Supporting the Knowledge Discovery Processes

Hesham A. Mansour[1], **Daniel Duchamp**[1], **and Carl-Arndt Krapp**[2]
[1]Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ 07030, USA
[2]FJA-US, Inc., 1040 Avenue of the Americas, 4th Floor, New York, NY 10018, USA

Abstract - *Knowledge Discovery in Databases (KDD) processes are complex, highly interactive and iterative. The similarities between KDD processes and software development processes suggest that approaches used to manage the development of software processes are also applicable and in fact advantageous to KDD processes. In this paper, we examine the current approaches for supporting KDD and note to their limitations in providing comprehensive and effective process support. We propose a language-based and process-oriented approach for supporting KDD processes that is based on explicitly representing KDD processes as process programs that can be analyzed, validated, and enacted. We illustrate the proposed approach using a novel process programming language that is designed to describe general process concepts as well as specific KDD concepts. Along with the KDD process language, an IDE-style development environment is proposed to assist in modeling and enacting KDD processes. The overall approach is evaluated and illustrated by modeling and enacting a traditional KDD process.*

Keywords: Data Mining, Knowledge Discovery in Databases, KDD Process, Process Programming.

1 Introduction

Today, KDD projects are typically approached in an unstructured, ad hoc manner [6]. The lack of systematic approaches for managing and keeping track of the different parts of KDD projects means that some steps may unintentionally be repeated, adding overhead to the knowledge discovery task. Rudiger et al. [6] have noted major problems during the development of many KDD projects at Daimler-Benz due to the lack of a methodology and lack of a usable process model with proper tool support. Marban et al. [28], [29] have noted that the number and complexity of data mining projects has increased in recent years, that nowadays there isn’t a formal process model for this kind of project, and that existing approaches are not correct or complete enough. They also noted that not all projects end successfully. The failure rate is actually as high as 60%.

In this paper, we proposes the Knowledge Discovery Process Modeling and Enacting Language (KDPMEL) along with its Process-Centered Software Environment (PCSE-KDD) that can be used to develop KDD processes in a way that is similar to developing software processes: KDD processes as process programs written in KDPMEL and exploited by PCSE-KDD to provide execution support and management for KDD processes.

Considerable value can be gained from materializing KDD processes via process programming. Novice participants, in particular, can benefit the most from knowing and learning their roles in the process and how their work and contributions would be coordinated and fit with others’ work and contributions. It has been observed by many KDD practitioners [22] that the results of KDD projects are often highly dependent on the experience of the persons doing the work. This phenomenon would likely be mitigated by having the work explicitly defined in a way that allows sharing the experience among the persons doing the work.

2 Current Approaches for Supporting KDD Processes and Their Limitations

We distinguish three major KDD support approaches found in the literature: activity-oriented support, KDD support environments, and process-oriented support.

2.1 KDD Activity-Oriented Support

This approach provides support only for individual activities such as data preprocessing or algorithm selection and settings. Examples of such support are proposed by [14]-[18], [42]. In this approach, the process concept, if used at all, is only represented in the form of documentation and guidelines. Also, the tools supporting the process tasks are isolated without any means of integration that would facilitate their usage and can enforce consistency conditions among the produced artifacts.

2.2 KDD Support Environments

The development of software environments supporting the overall KDD process has been identified by Padhraic [5] as a grand challenge for KDD. The architectures proposed for such environments are mainly based on a hardwired process model such as the traditional KDD process model [1] or the CRISP-DM [2] model. Some of the research efforts that fall under this category can be found in [8], [19]-[27], [30], [41].

Although these environments can be used to define and execute KDD processes, the provided process support is mainly derived from the hardwired process model, which includes major process phases along with their generic tasks and simple interactions. This sort of guidance is too generic and clearly insufficient for effectively supporting KDD processes, where specialized guidance is needed to assist in selecting valid, desirable, and effective process configurations. Moreover, the guidance provided by these systems is limited to a few standard KDD techniques and prescribed set of supporting tools that are mandated by the environments. This generic guidance and limited support is insufficient for a dynamic field such as KDD where scores of new techniques, guidelines, and tools for data manipulation and analysis are added on regular basis.

2.3 KDD Process-Oriented Support

KDD process-oriented support assures that activities performed within KDD processes are properly controlled and data analysis and manipulation techniques are used appropriately. Very few researchers [7], [22], [32]-[35] have addressed the issue of providing process-oriented support.

The project CITRUS [22] has extended a commercial knowledge discovery tool, CLEMENTINE, to enhance its user support capabilities by providing a process support interface. The main limitations of CITRUS are its dependency on CLEMENTINE and its supported process model and offered techniques; and its high-level process guidance.

Osterweil et al. [7] were the first to propose the use of process programming to address the coordination of KDD techniques. This process-oriented approach is illustrated using the Little-JIL language. Little-JIL is a visual language derived from a subset of JIL, a “process language” originally developed for software development processes [9]. Although the use of Little-JIL to specify a representative bivariate regression process has shown that many coordination aspects of the process can be easily expressed, it has uncovered some deficiencies in the language. Although this attempt is very promising, it concentrates on supporting only the coordination aspect of the process. In addition to the discovered deficiencies in Little-JIL, only the simplest processes can be modeled visually using Little-JIL.

Collaboration is another process-oriented aspect that has been recently adopted by some KDD support proposals [32]-[35], which are based on the paradigm of Service-Oriented Architecture (SOA). Collaborative KDD (CKDD) is an emerging field that seeks to cope with the distributed structure of modern organizations and the consequent increased complexity of the knowledge discovery process [31]. Although CKDD is beyond the scope of our work, it’s worth noting that our process-oriented approach, which employs a process-centered environment, inherently promotes the collaboration aspect.

2.4 The Need for a more Comprehensive Approach for Supporting KDD Processes

As discussed previously, the current state of KDD support is that the first approach (activity-oriented) supports only fragments of the KDD process, the second approach (KDD support environments) supports only a particular KDD process model, and the third approach (process-oriented) supports only certain process aspects of the KDD process.

Each of these half measures is inadequate. The support needed for a KDD process varies greatly based on the specifications of the concrete KDD process, and cannot be based purely on a generic process model. A KDD process might have many different configurations and can be instantiated in a number of ways, and each configuration might require different support.

Highly specialized KDD process support presently takes the form of technical documentation that specifies desirable and effective configurations for the process steps in an informal way [7]. This requires KDD practitioners to learn and apply these specifications manually. While this may be acceptable for experienced KDD practitioners who can cope with only high level guidance, it is not suitable for the less sophisticated KDD practitioners who participate in the development and enactment of the majority of KDD processes. We believe that the guidance necessary for the typical user can be achieved by explicitly representing the concrete KDD process using a flexible and rigorous formalism provided by the language-based approach of process programming. Further, explicit representation of the KDD process can be exploited by a process-aware environment to support process execution and guide users in carrying out their duties.

3 The Knowledge Discovery Process Modeling and Enacting Language (KDPMEL)

KDPMEL is a novel process programming language for modeling and enacting KDD and data analysis activities along with their resources, interactions, and coordination.

The process aspects of the language such as process structuring and task ordering are similar to those found in general process languages, such as JIL [9], Little-JIL [7], and PML [11]. The KDD aspects of the language are specific features for modeling KDD artifacts, tools, and tasks.

KDPMEL supports modeling KDD tasks at different levels of detail and abstraction in order to specify both generic and specialized tasks. Specialized KDD tasks are defined in KDPMEL through external commands that are modeled in the program and executed through a flexible plug-in mechanism for the tools of these commands.

KDPMEL provides special control constructs to explicitly model task dependencies on other tasks. This feature is particularly important for KDD and is intended to effectively manage the dependencies between KDD techniques. Having these dependencies explicitly defined can assure that they are appropriately handled.

A process program written in KDPMEL is organized into three major sections that specify the resources (*artifacts*, *roles*, and *tools*) of the process, general information about the process (*goal*, *input*, *outcome*, and *assessment*), and the steps (*activity*, *action*, and *command*) of the process along with their sequencing (*sequence*, *parallel*, *choice*, and *loop*) and dependencies (*disallow*, *require*, and *enable*).

3.1 Language Goals

The major goals of the KDPMEL are the *simplicity*, *flexibility*, *expressiveness*, and *generality*.

3.2 Language Approaches

KDPMEL combines both the graphical and process language modeling approaches. It employs a number of graphical modeling editors on top of a textual process programming language designed specifically for KDD. The goal of the modeling editors is to facilitate the construction and presentation of certain process components, as represented by four types of graphs that display the overall process (process graph), the resources of the process (resources graph), a graph for each activity (activity graph), and a graph for each action (action graph). Furthermore, a read-only graph that shows the progress of the artifacts within the process (artifact flow graph) is provided. The process graph indicates process information such as goal, input, outcome, etc. The resources graph shows the artifacts, tools, and roles/actors of the process. Each activity graph shows the activity's constituent actions while each action graph provides information such as tools utilized by the action, the artifacts consumed and produced by the action, and the actor assigned to the action. In addition to the graphical editors, a number of form-based editors exist (process form, activity form, artifact form, role/actor form, and tool form) to present and update certain information that is best shown and updated in a form-based style.

Combining different types of editors and views in source-based, graph-based, and form-based styles is novel and allows both technical and non-technical users to participate in the modeling phase of the process. This hybrid modeling approach combines the benefits of the underlying approaches and enables specification of high-level process models as well as more complex ones in a manner that is convenient for both technical and non-technical users.

3.3 KDPMEL Meta-Model (Abstract Syntax)

KDPMEL is defined in terms of a meta-model [38] based on the OMG's SPEM [13] and CWM [37] meta-models, which represents the abstract syntax and static semantics of the language. The KDPMEL meta-model consists of a Core meta-model upon which the other meta-models depend, a Process meta-model representing the process aspects, and a KDD meta-model representing the KDD aspects.

3.4 KDPMEL Concrete Syntax

The concrete syntax of KDPMEL is provided in two flavors, textual and graphical, to serve different purposes. The textual concrete syntax is useful when specific complex details must be specified. The graphical concrete syntax is easy to understand and use, and is useful for communicating structural and higher level views of the process. Also, a form-based interface is provided for process components to allow for presenting and updating their properties.

3.4.1 KDPMEL Textual Concrete Syntax (Grammar)

The Process meta-model provides process specific entities such as *Process*, *Activity*, and *Action*. The syntax for defining these constructs is given by the following rules:

<process> ::= "**process**" <IDENTIFIER> "{"…"}"
<activity> ::= "**activity**" <IDENTIFIER> "{"…"}"
<action> ::= "**action**" <IDENTIFIER> "{"…"}"

Process Syntax

A process can be decomposed into an ordered collection of activities. The activities can be grouped using one of the control constructs *sequence*, *parallel*, *choice*, or *loop*. The process syntax is defined as follows:

```
<process> ::= "process" <IDENTIFIER> "{" …
                    (<activitySequencing> | <activity>)*
               "}"
```

Activity Syntax

An activity represents a composite task and it is mainly intended to represent the phases of the KDD process. An activity may have pre-conditions and post-conditions to guard entry into and exit from the activity. An activity may consume and produce some artifacts during its performance, which is monitored by an actor. An activity can be decomposed into smaller units of sub-activities and/or actions. The activity syntax is defined as follows:

```
<activity> ::= "activity" <IDENTIFIER> "{"
                  ["preconds" <constraint> ("," <constraint>)*]
                  ["postconds" <constraint> ("," <constraint>)*]
                  [<consumedArtifacts>] [<producedArtifacts>]
                  [<performer>]
                  ["sub-activities" "{" (<activity>)+ "}"]
                      (<actionSequencing> | <action>)*
               "}"
```

Actions within an activity can be grouped using one of the control constructs *sequence*, *parallel*, *choice*, or *loop*. The following defines an activity that has two actions grouped by the *parallel* construct:

```
activity DataMining {
    parallel predict {
      action buildDecisionTreeModel {…}
      action buildNeuralNetModel {…}
    }
}
```

The decomposition of an activity allows for defining the tasks of the activity. The activity→sub-activity decomposition provides a strict control, whereas the activity→action decomposition provides both strict (e.g., *sequence*/*loop*) and loose control (e.g., *choice*/*parallel*).

Action Syntax

An action represents a primitive task and it is intended to represent the generic tasks of the KDD process. An action may have pre-conditions and post-conditions. An action is performed by an actor with the help of some tools. An action may consume and produce some artifacts. To help perform an action, guidance information for the actor may be associated with the action. Finally, an action may have dependencies with other actions. The action syntax is defined as follows:

```
<action> ::= "action" <IDENTIFIER> "{"
                 ["preconds" <constraint> ("," <constraint>)*]
                 ["postconds" <constraint> ("," <constraint>)*]
                 [<consumedArtifacts>] [<producedArtifacts>]
                 [<performer>] [<utilizedTools>]
                 [<dependencyDecl>] [<guidanceDecl>]
             "}"
```

The following example is a KDD task for building a decision tree model that specifies that the task is performed by a data mining analyst with the help of a particular mining tool over a specific dataset:

```
action buildDecisionTreeModel {
   consume sampleDataset;
   produce sampleDecisionTreeModel;
   performer dmAnalyst; utilize { call miningTool }
}
```

3.4.2 KDPMEL Graphical Syntax

The *Process*, *Activity*, and *Action* constructs, in addition to the process resources are represented in the graphical syntax. In addition to the graph-based notation, a form-based interface is provided. The source-based, graph-based, and form-based notations of the process program share a common object model for the process that is updated by and translated into the various representations of the process program.

3.5 KDPMEL Semantics

3.5.1 Control Flow and Ordering

The activities within a process and the actions within an activity can be grouped using one of the control constructs *sequence*, *parallel*, *choice*, or *loop*. The default grouping is *sequence*. Additionally, activities may be decomposed into a hierarchy of sub-activities and actions.

3.5.2 Dependency Control Constructs

The states of KDPMEL tasks and their dependency requirements are recorded during the execution of the tasks. Upon beginning the execution of a task, a test is performed against the completed actions to check whether their *disallow* dependency prohibit execution of the task. The *enable* dependency is checked only for the *choice* control construct to determine if one of the choices has been enabled by a completed action. If that was the case, the actor making the choice will be notified. Another test is performed upon beginning the execution of an action to check its *require* dependency against the completed tasks to determine if the action can be executed. An action can only be executed if its required tasks are completed.

We believe that this is a novel approach for managing KDD task dependencies that are dynamically reflected at runtime, as opposed to statically structuring these tasks according to their dependencies at modeling time [7], which provides more flexibility not only in modeling time but in execution time also. In addition, it also leads to much shorter programs.

3.5.3 Action Specialized KDD Tasks

KDPMEL models specialized KDD tasks through the use of external commands that can be associated with an action and a tool. Each tool that is associated with an external command is represented by a plug-in module that is invoked to execute the command.

3.5.4 Task States and Transitions

The states and transitions of KDPMEL tasks are implemented using the State Pattern [40]. Tasks within a KDPMEL program go through several states during the execution of the program. The state of a task changes based on the control flow of the program, the availability of the resources needed by a task, and the explicit response from human actors. KDPMEL adopts states similar to those of Little-JIL--*posted*, *started*, *completed*, *terminated*, and *retracted*--- and adds the two new states *suspended* and *resumed* that have been suggested by Lee [12]. Figure 1 illustrates KDPMEL task states and their transitions as suggested in [12].

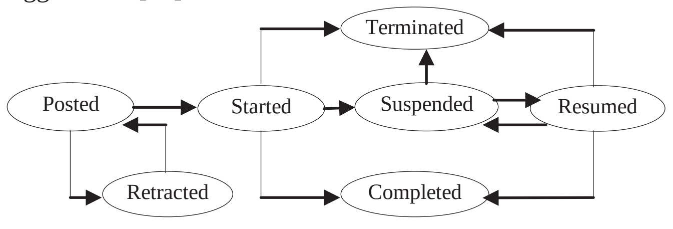


Fig. 1. Task State Transitions in KDPMEL [12]

When a KDPMEL task becomes available for execution, a task instance is created and its state is set to *posted*. A *posted* task instance is started by an explicit *start* action from the task performer (actor), which sends a start event to the task controller to change the state of the task instance to *started*. A *posted* task instance can also be temporarily retracted by a *retract* action. A *started* task instance is completed by an explicit *complete* action. A *completed* task indicates that the task has successfully finished execution. This causes the enactment engine to continue executing the rest of the program by finding and posting the next available task. A *started* task instance can also be terminated by an explicit *terminate* action. A *terminated* task indicates an exception that caused the task not to be finished successfully. The handling of a terminated task varies depending on the type of control construct governing the task. For instance, while a *terminated* task in a *sequence* or *loop* control construct causes the termination of the other tasks in the construct, a *terminated* task in a *choice* control construct causes the enactment engine to offer the construct alternatives to the actor to select a task.

4 The KDD Process-Centered Support Environment (PCSE-KDD)

PCSE-KDD is an Integrated Development Environment that is built around KDPMEL, with an IDE-style approach to facilitate the development, execution, and management of KDPMEL programs. The environment offers a variety of services, similar to those offered by PCSEEs, but directed toward KDD processes.

4.1 Architecture

The environment implements the process definition/instantiation/enactment paradigm, found in PCSEEs, and includes a number of modeling editors for modeling KDD processes, an Enactment Engine for providing runtime process execution support, and a Repository for providing persistency support to both process artifacts and process execution states. Figure 2 illustrates the high level architecture of the environment.

The PUI exposes the various services offered by the environment. Through the PUI, users are able to define, update, and persist process models during the modeling phase, instantiate a process model for enactment, participate in the enactment phase by performing interactive tasks in the process, are notified by the enactment engine about the status of the process being enacted, and are guided by the enactment engine about what to do next.

The Enactment Engine is responsible for executing KDPMEL programs. It guides and supports users in performing their assigned tasks, controls the invocation of tools, accesses the process artifacts, and maintains the process execution states. It includes three significant components: KDPMEL Interpreter, the Repository Management Unit (RMU), and the Tool Invocation Unit (TIU). The KDPMEL Interpreter implements the semantics of the language. The RMU maintains the process data. The TIU manages the invocation of tools specified in the process program.

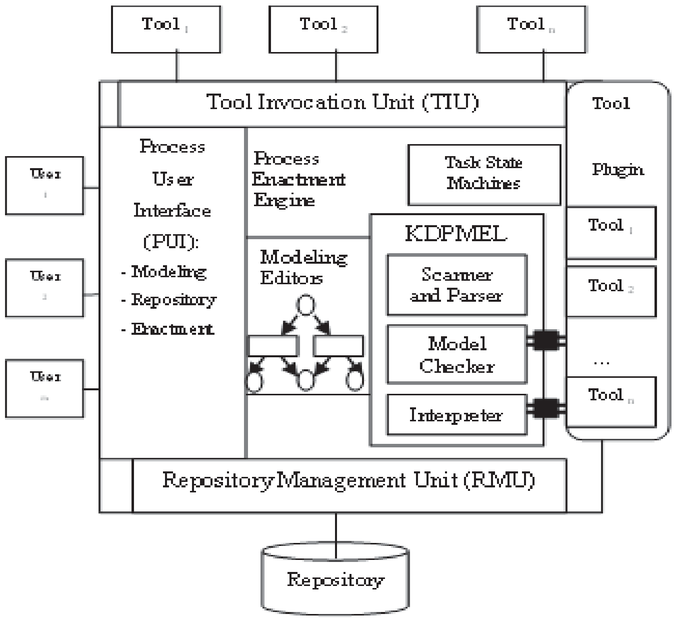


Fig. 2. The high level architecture of the PCSE-KDD

5 An Example for Developing a Traditional KDD Process in PCSE-KDD

The example process is used for predicting the likelihood that bank customers will reply to a mailing campaign for buying a Personal Equity Plan (PEP) [36].

5.1 The Example Process Specification

Data Selection

The data is available in a comma-separated value (CSV) file.

Data Preparation

This stage includes steps to transform the selected dataset file into its WEKA dataset representation, remove unnecessary attributes, and construct the training and test datasets.

Data Mining

A decision-tree technique using the C4.5 (J48) WEKA classifier [39] is used to predict the PEP value (YES/NO).

Interpretation/Evaluation

The results are evaluated using a tree visualization technique to display the decision-tree graph model in addition to inspecting the detailed results using a text editor.

5.2 The KDPMEL Prediction Process Program

5.2.1 Process Resources

Figure 3 illustrates the process resources as they are depicted in the Resources Graph.

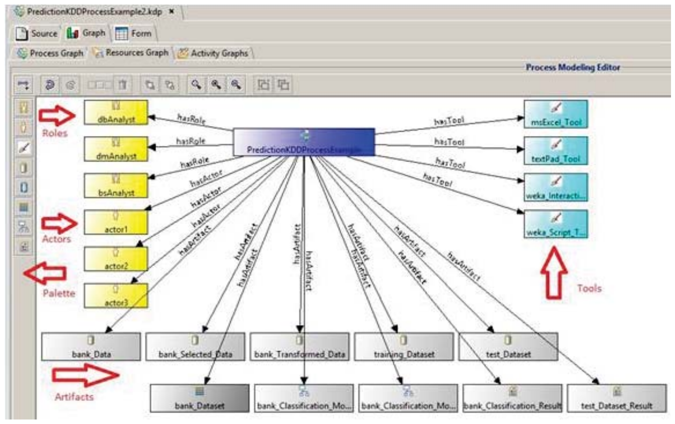


Fig. 3. The Example Process Resources Graph

Artifacts

Figure 4 illustrates the process artifacts as they are depicted in the Artifact Flow Graph.

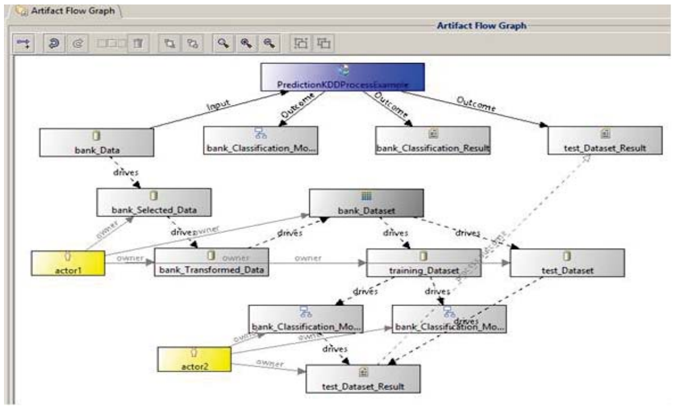
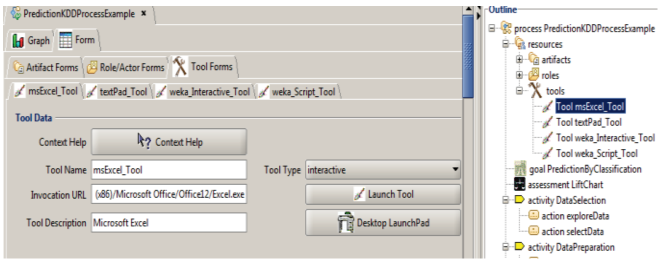


Fig. 4. The Example Process Artifact Flow Graph

Tools

The tools utilized by the process are Microsoft Excel for the data selection tasks, the *WEKA* data mining framework [39] in both the interactive and command-line modes for the Data Preparation, Data Mining, and Interpretation tasks, and the text editor *TextPad* for some of the Interpretation tasks. Figure 5 illustrates the utilized tools.



Tool Data

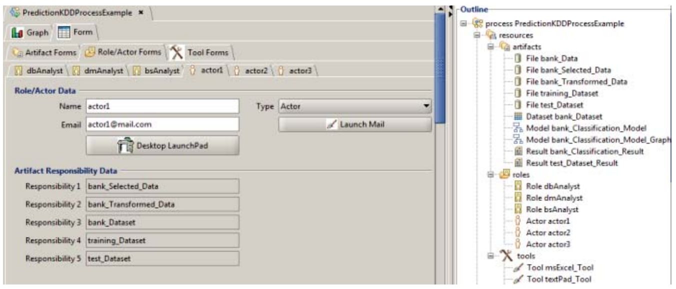
Tool Name	msExcel_Tool	Tool Type	interactive
Invocation URL	(x86)/Microsoft Office/Office12/Excel.exe		
Tool Description	Microsoft Excel		

```
process PredictionKDDProcessExample
  resources
    artifacts
    roles
    tools
      Tool msExcel_Tool
      Tool textPad_Tool
      Tool weka_Interactive_Tool
      Tool weka_Script_Tool
  goal PredictionByClassification
  assessment LiftChart
  activity DataSelection
    action exploreData
    action selectData
  activity DataPreparation
```

Fig. 5. The Example Process Utilized Tools

Roles/Actors

Three different roles--database, data mining, and business analysts--are fulfilled by three different actors are defined in the program. Figure 6 illustrates the process roles/actors.



Role/Actor Data

Name	actor1	Type	Actor
Email	actor1@mail.com		

Artifact Responsibility Data

Responsibility 1	bank_Selected_Data
Responsibility 2	bank_Transformed_Data
Responsibility 3	bank_Dataset
Responsibility 4	training_Dataset
Responsibility 5	test_Dataset

```
process PredictionKDDProcessExample
  resources
    artifacts
      File bank_Data
      File bank_Selected_Data
      File bank_Transformed_Data
      File training_Dataset
      File test_Dataset
      Dataset bank_Dataset
      Model bank_Classification_Model
      Model bank_Classification_Model_Graph
      Result bank_Classification_Result
      Result test_Dataset_Result
    roles
      Role dbAnalyst
      Role dmAnalyst
      Role bsAnalyst
      Actor actor1
      Actor actor2
      Actor actor3
    tools
      Tool msExcel_Tool
      Tool textPad_Tool
```

Fig. 6. The Example Process Roles/Actors

5.2.2 The Process Phases

The process includes four phases for *data selection*, *data preparation*, *data mining*, and *interpretation*. Each phase is defined using a KDPMEL *activity* construct as follows:

```
process PredictionKDDProcessExample { …
  activity DataSelection {...}
  activity DataPreparation {...}
  activity DataMining {...}
  activity Interpretation {...}
}
```

Figure 7 illustrates the phases of the process as they are depicted in the Process Graph.

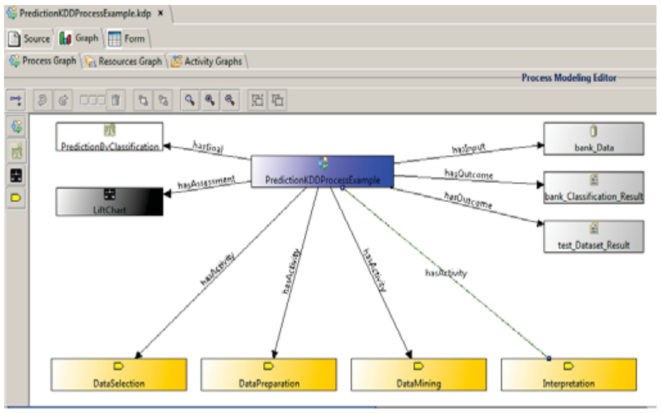


Fig. 7. The Process Graph of the Example Process

Each phase includes a number of generic and specialized tasks that are defined using KDPMEL *action* and *command* constructs. For instance, the *DataPreparation* phase is defined in KDPMEL as follows:

```
activity DataPreparation {
  action transformCsvData { ...
    command transformCSVDataCommand {...}  ...
  }
  action viewTransformedCsvData {...}
  action constructMainDataset { ...
    command transformCSVDataCommand {...} ...
  }
  action viewConstructedDataset {...}
  action buildTrainingAndTestDatasets { ...
    command buildTrainingDatasetCommand {...}
    command buildTestDatasetCommand {...}  ...
  }
}
```

The *DataPreparation* activity (Figure 8) includes a sequence of five tasks for transforming the CSV data to its *WEKA* representation, viewing the transformed data, constructing the main dataset, viewing the constructed dataset, and constructing the training and test datasets.

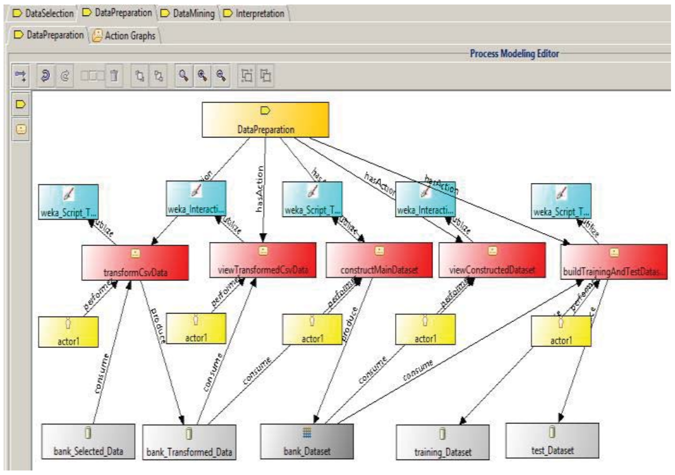


Fig. 8. The Example Process *DataPreparation* Activity Graph

5.2.3 The Process Actions and Commands

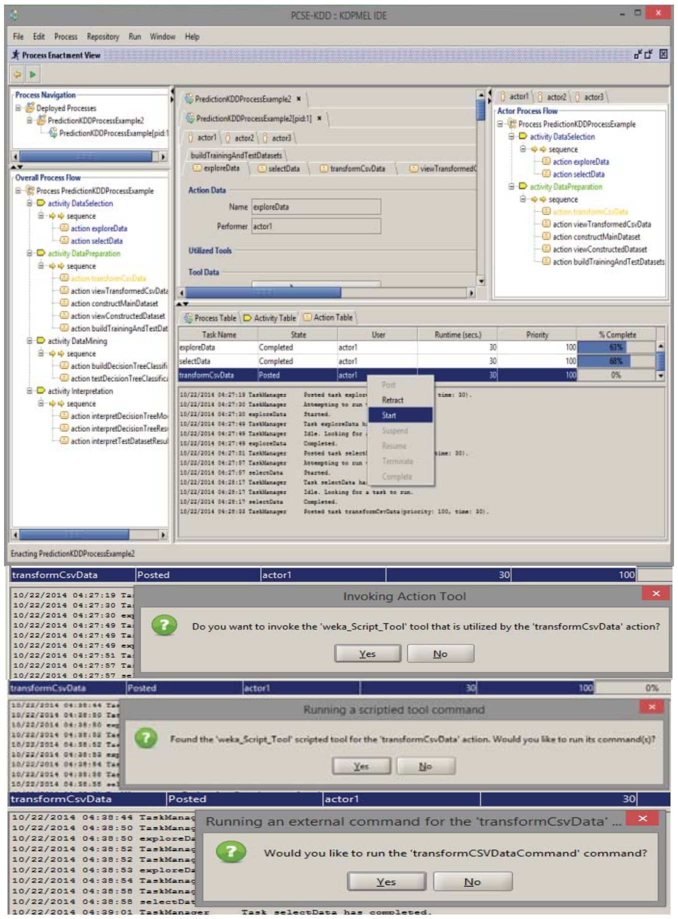
The process includes a number of generic and specialized tasks. For instance, the *DataPreparation* activity includes a specialized task that is used to transform the CSV data file to its *WEKA* format. This is defined as follows:

```
action transformCsvData { …
  utilize {
    call weka_Script_Tool {
      command transformCSVDataCommand { …
        input bank_Selected_Data; output …
        operation "weka.core.converters.CSVLoader";
        parameters "";
      }
    }
  }
}
```

5.3 Enacting the Example Process Program

The PCSE-KDD Enactment Engine establishes a process instance and presents it in the Enactment Perspective. The Enactment Perspective displays the overall process execution flow organized by the actors. Each actor is presented with its assigned tasks in a tree view. The description of each task is presented in a form view. A GUI mechanism to apply appropriate transition states (e.g., a start command to execute a posted task) for each task is provided to the actor.

Figure 9 illustrates the enactment of the *transformCsvData* action. When selecting the start command, the Enactment Engine starts the action and identifies its utilized tool and specialized command and offers to invoke them through dialogs. The actor confirms the invocation.



Task Name	State	User	Runtime (secs.)	Priority	% Complete
exploreData	Completed	actor1	30	100	63%
selectData	Completed	actor1	30	100	60%
transformCsvData	Posted	actor1	30	100	0%

```
10/22/2014 04:28:17 TaskManager    Idle. Looking for a task to run.
10/22/2014 04:28:17 selectData     Completed.
10/22/2014 04:28:33 TaskManager    Posted task transformCsvData(priority: 100, time: 30).
```

Enacting PredictionKDDProcessExample2

Invoking Action Tool: Do you want to invoke the 'weka_Script_Tool' tool that is utilized by the 'transformCsvData' action?

Running a scriptied tool command: Found the 'weka_Script_Tool' scripted tool for the 'transformCsvData' action. Would you like to run its command(s)?

Running an external command for the 'transformCsvData' ...: Would you like to run the 'transformCSVDataCommand' command?

```
10/22/2014 04:39:01 TaskManager    Task selectData has completed.
```

Fig. 9. The *transformCsvData* Action execution dialogs

5.4 Evaluation and Lessons Learned

Our experience using KDPMEL to specify the example KDD process, as well as other KDD processes, supports our first hypothesis that a language-based and process-oriented approach is a flexible and effective approach to precisely and explicitly specify KDD processes as process programs that can be manipulated by programming techniques to reason about the process and support its correct execution.

KDPMEL simplicity goal is achieved by having simple syntax. KDPMEL flexibility goal is mainly achieved by providing various levels for representing tasks at different levels of detail. The generality goal is mainly achieved by not making KDPMEL and PCSE-KDD bound to any particular process models, techniques, or tools. KDPMEL expressiveness goal is achieved by providing constructs to represent both generic and specialized tasks along with their sequencing and dependencies, consumed and/or produced artifacts, utilized tools, and performing actors.

Our experience using KDPMEL and PCSE-KDD to represent and execute the example process supports our second hypothesis that effective support and customized guidance, which depend on the concrete process itself rather than its generic process model, can be achieved by manipulating the explicit representation of the process in order to manage its various components and support its performance.

6 Conclusions

KDPMEL provides a hybrid modeling approach for specifying KDD processes, mixing different types of editors and views in source-based, graph-based, and form-based styles to allow both technical and non-technical users to participate in the development of KDD processes.

PCSE-KDD is an Integrated Development Environment for KDPMEL. It has been prototyped in Java plus a number of open source libraries and tools. It has the look and feel of Eclipse IDE and has a similar Workbench that includes three different Perspectives, similar to Eclipse's Perspectives, for its Modeling, Enactment, and Management functionalities.

In PCSE-KDD/KDPMEL the process concept is supported and enforced according to a specialized KDD process that includes specific tasks organized according to their sequencing, dependencies, and alternatives. In PCSE-KDD, tools are loosely integrated through a flexible and expandable plug-in mechanism. They are launched automatically and dynamically according to the execution order of the process tasks. PCSE-KDD provides a centralized repository for maintaining and managing the process artifacts. PCSE-KDD employs an engineering approach to develop KDD processes. It is a language-based and process-driven approach. The process is explicitly defined as a program in KDPMEL and manipulated by PCSE-KDD to support its analysis and execution. Specialized user guidance during execution is extracted from the interpretation of the process program. Users are offered their assigned tasks and supported in executing them. In this language-based approach, KDD processes are managed. Their specifications can evolve and executions can be repeated. Moreover, they are validated according to standard programming techniques.

Our future work includes modeling a wide range of KDD processes and increase the level of sophistication of those processes, expanding the support for more detailed KDD artifacts, and continuing the development of KDPMEL and PCSE-KDD to provide more enhanced and expanded graphical modeling to cover the entire process, better user interaction during process enactment, and to expand the integration with a wider range of external process resources.

7 References

[1] Fayyad U. M., Piatetsky-Shapiro, G., and P. Smyth. "Knowledge Discovery and Data Mining: Towards a Unifying Framework". MIT press, Cambridge, Mass., 1996.

[2] Colin Shearer. "The CRISP-DM Model: The New Blueprint for Data Mining". Journal of Data Warehousing, Volume 5, Number 4, 2000.

[3] Graham J. Williams and Zhexue Huang. "Modeling the KDD Process". 1996.

[4] SK Gupta, Vasudha Bhatnagar, and SK Wasan. "A Proposal for Data Mining Management System". Dept. of Computer Science and Engineering, Indian Institute of Technology, 2001.

[5] Padhraic Smyth. "Breaking Out of the Black-Box: Research Challenges in Data Mining". The 2001 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2001.

[6] Rudiger Wirth and Jochen Hipp. "CRISP-DM: Towards a Standard Process Model for Data Mining". DaimlerChrysler Research & Technology.

[7] David Jensen, Yulin Dong, Barbara S. Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton, Jr., and Alexander Wise. "Coordinating Agent Activities in Knowledge Discovery Processes". Department of Computer Science, University of Massachusetts Amherst, 1999.

[8] C. Zeng, Y. Jiang, L. Zheng, J. Li, L. Li, H. Li, C. Shen, W. Zhou, T. Li, B. Duan, M. Lei, and P. Wang. "FIU-Miner: A Fast, Integrated, and User-Friendly System for Data Mining in Distributed Environment". In Proc. of the Nineteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013.

[9] Sutton Jr, Stanley M., and Leon J. Osterweil. "The design of a next-generation process language". In Software Engineering—ESEC/FSE'97, pp. 142-158. Springer Berlin Heidelberg, 1997.

[10] Fayyad U. M., Piatetsky-Shapiro, G., and Uthurusamy, R. "Summary from the KDD-03 Panel – Data Mining: The Next 10 Years". The 9th International Conference on Data Mining and Knowledge Discovery (KDD-03), 2003.

[11] Noll, J. and Scacchi, W. "Specifying process-oriented hypertext for organizational computing". Journal of Network and Computer Applications 24, 39-61, 2001.

[12] Lee, H. "Evaluation of Little-JIL 1.0 with ISPW-6 Software Process Example". Department of Computer Science, University of Massachusetts, Amherst, March 1999.

[13] OMG, Inc. "Software Process Engineering Metamodel Specification".URL:http://www.omg.org/technology/documents/formal/spem.htm, Version 1.1, January, 2005.

[14] Abraham Bernstein, Foster Provost, Shawndra Hill. "Toward Intelligent Assistance for a Data Mining Process: An Ontology-Based Approach for Cost-Sensitive Classification". IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 4, pp. 503-518, April, 2005.

[15] Robert Engles, Guido Linder, and Rudi Studer. "A Methodology for Providing User Support for Developing Knowledge Discovery Applications". URL:http://www.aifb.uni-karlsruhe.de/WBS/publications/

[16] Blake, M. B. and Williams, A. B. "Development and Operational Processes for Agent-Oriented Database Navigation for Knowledge Discovery". In Proc. of the 15th International Conference on Software Engineering & Knowledge Engineering (SEKE '2003), 2003.

[17] Petr Aubrecht, Petr Miksovsky, and Lubos Kral. "SumatraTT: a Generic Data Pre-processing System". 14th

International Workshop on Database and Expert Systems Applications (DEXA'03), 2004.

[18] Kamil Matousek and Petr Aubrecht. "Data Modeling and Pre-processing for Efficient Data Mining in Cardiology". IEEE ITAB'06, Ioannina, October 28, 2006.

[19] S.K. Gupta, V.Bhatnagar, S.K. Wasan, and DVLN Somayajulu. "Intension Mining: A New Paradigm in Knowledge Discovery". Dept. of Computer Science and Engineering, Indian Institute of Technology, 2000.

[20] M.C. Fernandex, O. Delgado, J. I. Lopez, M. A. Luna, et al. "DAMISYS: An Overview". In Proc. of 1st Int'l Conf. on Data Warehousing and Knowledge Discovery, Aug 1999.

[21] R. Meo, G. Psaila, and S. Ceri. "A Tightly-Coupled Architecture for Data Mining". In Proc. of 1st Int'l Conf. on Data Warehousing and Knowledge Discovery, Aug 1999.

[22] Rudiger Wirth et al. "Towards Process-Oriented Tool Support for Knowledge Discovery in Databases". DaimlerChrysler Research & Technology, 1997.

[23] S.K. Gupta, V.Bhatnagar, and SK Wasan. "Architecture for knowledge discovery and knowledge management". Knowledge and Information Systems, Volume 7, Issue 3, pp. 310-336, 2005.

[24] Martin Spott and Detelf Nauck. "Intelligent Data Analysis: Developing New Methodologies Through Pattern Discovery and Recovery". Chapter I: Automatic Intelligent Data Analysis, Copyright ©, 2009, IGI Global, ISBN: 978-1-59904-982-3, 2008.

[25] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. "YALE: Rapid Prototyping for Complex Data Mining Tasks". In Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.

[26] J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, and F. Herrera. "KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems". Soft Computing 13:3 (2009) 307-318, 2009.

[27] A. Romei, S. Ruggieri, and F. Turini. "KDDML: a middleware language and system for knowledge discovery in databases". In Data and Knowledge Engineering. Vol 57, Issue 2, pages 179-220, May 2006.

[28] Marban, O., Mariscal, G., Menasalvas, E., and Segovia, J. "An Engineering Approach to Data Mining Projects". Intelligent Data Engineering and Automated Learning – IDEAL 2007, LNCS 4881, pp. 578-588, 2007.

[29] Marban, O., Segovia, J., Menasalvas, E., and Fernndex-Baizn, C. "Toward data mining engineering: A software engineering approach". Information Systems 34 (1), 2009.

[30] Vincenzo Cannella, Giuseppe Russo, Daniele Peri, Roberto Pirrone, and Edoardo Ardizzone. "Towards MKDA: A Knowledge Discovery Assistant for Researches in Medicine". In Proc. of the 10th Conf. of the Italian Association for Artificial Intelligence, pp. 773-780, 2007.

[31] Diamantini, C., Potena, D., and Smari, W. "Collaborative Knowledge Discovery in Databases: A Knowledge Exchange Perspective". AAAI 2006 Fall Symposium on Semantic Web for Collaborative Knowledge Acquisition, 2006.

[32] Diamantini, C., Potena, D, and Panti, M. "Developing an open knowledge discovery support system for a network environment". In Proc. of the IEEE International Symposium on Collaborative Technologies and Systems, pages 274-281, Saint Louis, MO, USA, May 18-19, 2005.

[33] Diamantini, C. and Potena, D. "Representing Service Information in a Collaborative KDD Environment". In Proc. of the International Symposium on Collaborative Technologies and Systems, pages 331-338, Irvine, CA, USA, May 19-23, 2008.

[34] Diamantini, C., Potena, D., and Storti, E. "Collaborative management of a repository of KDD processes". International Journal of Metadata, Semantics and Ontologies, 9(4), 299-311, 2014.

[35] Esmin, A. A., Pereira, D. A., Pereira, M. R., & Araújo, D. L. "SMINER–a platform for data mining based on service-oriented architecture". International Journal of Business Intelligence and Data Mining, 8(1), 1-18, 2013.

[36] DePaul University, Chicago, IL, Classification via Decision Trees in WEKA. URL: http://maya.cs.depaul.edu/classes/ect584/WEKA/classify.html

[37] OMG, Inc., Common Warehouse Metamodel (CWM) Specification, URL: http://www.omg.org/technology/documents/formal/cwm.htm, Version 1.1, March 2003.

[38] Greg Nordstrom, Janos Sztipanovits, Gabor Karsai, and Akos Ledeczi. "Metamodeling - Rapid design and evolution of domain-specific modeling environments". IEEE Engineering of Computer Based Systems (ECBS), Nashville, TN, pp. 68-74, April 1999.

[39] University of Waikato, New Zealand. "Weka 3: Data Mining Software in Java". URL: http://www.cs.waikato.ac.nz/ml/weka/. Version 3.6, 2010.

[40] Open Source, The State Machine Compiler (SMC) Framework, URL: http://smc.sourceforge.net/

[41] Nurdatillah Hasim and Norhaidah Abu Haris. "A study of open-source data mining tools for forecasting". In Proc. of the 9th International Conference on Ubiquitous Information Management and Communication (IMCOM '15), 2015.

[42] Serban, F., Vanschoren, J., Kietz, J.-U., and Bernstein, A. "A survey of intelligent assistants for data analysis". ACM Computing Surveys (CSUR) v.45 n.3, p.1-35, June 2013.