PCSE-KDD: A Process-Centered Support Environment for the Knowledge Discovery Processes

Hesham A. Mansour
FJA-US, Inc., 1040 Avenue of the Americas, 4th Floor, New York, NY 10018, USA

Abstract – *Current support for Knowledge Discovery in Databases (KDD) is provided only for fragments of the process, a particular KDD process model, or most recently certain process aspects. 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. There is a need for a more comprehensive support approach that can cover the entire process, target concrete process specifications, and include various aspects of the process. KDD processes are similar to software processes and they can benefit from advancement of software engineering and process technology to facilitate their development, support their execution, and ultimately improve their effectiveness, utilization, and outcomes. This paper proposes the Process-Centered Support Environment for KDD (PCSE-KDD) processes that is based on explicitly representing these processes as process programs that can be developed, managed, and enacted by the environment. This approach has been successfully used to provide support for developing software processes and we propose to transplant this approach into the KDD field. With the proposed approach, KDD processes can be flexibly captured at different levels of details in a clear, precise, and explicit way that can enable reasoning about the process, insuring its correct execution, and supporting its performance.*

Keywords: KDD Process, Process Programming, Process-Centered Support Environments

1 Introduction

Although KDD is now widely accepted as a complex process with many different phases and non-trivial interactions, little support is provided to the various steps of the process or to manage the overall process.

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. [18] 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. The result is wasted resources and unnecessarily long development times, in addition to the fact that the results were highly dependent on the experience of the persons doing the work. Marban et al. [8] 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%. The intrinsic features of the KDD process, together with the main difficulties in its application, make the development and management of a KDD application, particularly of exploratory nature, very complex [19].

Current support for KDD is provided only for fragments of the process (*activity-oriented* support), a particular KDD process model (*KDD support environments*), or most recently certain process aspects (*process-oriented* support), such as the coordination or collaboration [1]. In the activity-oriented support 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. The process support provided by most existing KDD support environments is mainly derived from a hardwired generic KDD 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 tool 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. Among the very few proposals that apply process-oriented support to KDD, only [4] uses a process language approach based on Little-JIL to explicitly represent and support only the coordination aspect of KDD processes. In addition to the discovered deficiencies in Little-JIL, only the simplest processes can be modeled visually using Little-JIL. For additional information about the different approaches for supporting KDD processes and their limitations, see [1].

The recognition that software processes can themselves be described as software is attributed to Osterweil [20], and has led to the development of process programming as part of software engineering, as well as ongoing research into process-centered environments. The idea of using a Process

ISBN: 1-60132-431-6, CSREA Press ©

Modeling Language (PML) to encode a software process as a "process model", and enacting this using a process-sensitive environment is now well established [21]. Process-Centered Software Engineering Environments (PCSEEs) form the most recent generation of environments supporting software development activities [22]. They aim to support software development activities by exploiting an explicit representation of the software process---a process program---that specifies how to carry out the process activities and how to use and control the process supporting tools.

Although some researchers [4], [7]-[9] have recognized the similarities between KDD processes and software development processes, none (to our knowledge) has proposed a comprehensive approach for developing KDD processes through a Process-Centered KDD Support Environment based on PCSEEs to enable and facilitate the modeling, execution, and management of KDD processes.

In this paper, we propose the Process-Centered Support Environment for KDD (PCSE-KDD) for modeling, enacting, and managing KDD processes. The environment aims to provide effective management for the KDD process by supporting its entire lifecycle, and offering a variety of services, similar to those offered by PCSEEs, but directed toward KDD processes. Environment support includes assistance for process developers, maintenance of process resources, automation of routine tasks, invocation and control of development tools, and enforcement of mandatory rules and practices. The environment implements the process definition/instantiation/enactment paradigm found in PCSEEs and is based on the KDD process programming language KDPMEL [1], [2]. The environment 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.

Achieving a general-purpose data mining and knowledge discovery support environment is an undertaking that has been described to be quite a challenging problem in [23] and was predicted in 2003 to be among the most important KDD issues that will not show any measurable and notable scientific progress in the next 10 years [24]. This pessimism was mainly because of the complex nature of the KDD process and its branching factors in terms of selecting specific methods and supporting tools, branching which has caused commercial data mining products to be limited to a few standard techniques and to provide guidance based only on a hardwired process model. In contrast, we demonstrate that a general-purpose KDD support environment can be achieved by separating KDD process definitions from the environment and by providing the appropriate mechanisms for integrating these definitions with the environment. This separation of concerns can achieve significant flexibility in supporting a wide range of process specifications that can evolve over time and generality due to the fact that the environment is not bound to any particular KDD process models, techniques, or tools. Process technology can provide the appropriate approaches for achieving this separation of concerns. The paper is structured as follows. This section provides background information and the motivation for our work. Section 2 presents the Process-Centered Support Environment for KDD (PCSE-KDD) and illustrates its major components. Section 3 outlines the implementation details of PCSE-KDD. Section 4 concludes the paper and outlines future work.

1.1 Process-Centered Software Engineering Environments (PCSEEs)

PCSEEs address three distinguishable domains: the modeling, enactment, and performance domains. The modeling domain comprises all activities for defining and maintaining process models using a formal language with an underlying operational semantics that enables mechanical interpretation of the models. The enactment domain encompasses what takes place in the environment to mechanically interpret the process model by a so-called process engine. The performance domain is defined as the set of actual activities conducted by human agents and nonhuman agents (computers) during process execution. Process support provided by PCSEEs can be characterized by the typical interactions between the three domains (Fig. 1) [10]:

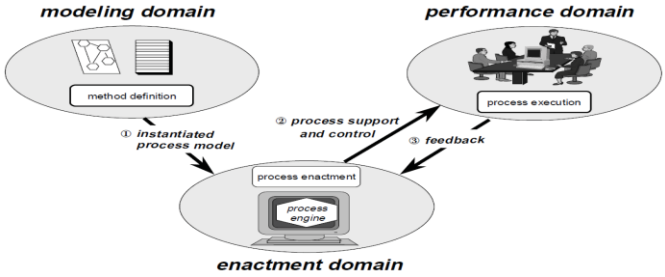

Fig. 1. Three domains of software process support [10]

1.2 KDD Processes and Software Development Processes

KDD processes are on one hand similar to software processes and on the other hand are different from software processes. The similarities between KDD processes and software processes suggest that approaches used to support the development of software processes, such as PMLs and related PCSEEs, are also applicable to KDD processes. However, because of the differences, some adaptation is needed in order to apply these approaches to KDD processes. Instead of adapting current KDD processes to match software processes as proposed in [8], [25], we propose to adapt these approaches to suit KDD processes in order to support their specific activities, techniques, components, and developers. This will provide support for current KDD processes as they are normally known by KDD practitioners who are not necessarily experts in software engineering. In addition, this will not force fundamental changes and additional activities on KDD processes and at the same time will not prevent form doing so when needed.

The approach that we propose to establish formal process models and methodologies for developing KDD processes is based on transplanting the idea that has been successfully used in software engineering to support the development of software processes into the KDD field. By transplanting this idea to KDD, we believe that we can formally and explicitly define KDD processes and provide a systematic methodology for their development and execution.

2 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. 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. KDPMEL provides various language constructs to control task sequencing and dependencies as well artifacts consumed and/or produced, tools utilized, and the actors performing the tasks. KDPMEL allows for capturing the process tasks at different levels of abstraction to represent the process phases (lifecycle) along with its generic and specialized KDD tasks. For additional information about KDPMEL, see [1], [2].

2.1 Architecture

Fig. 2 illustrates the high level architecture of the environment.

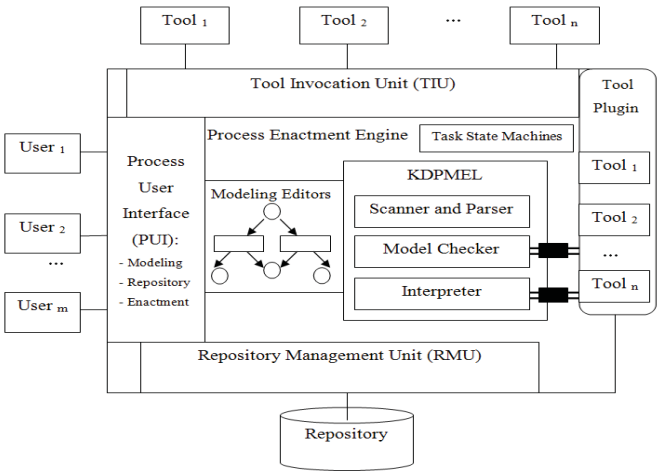

Fig. 2. The high level architecture of the PCSE-KDD

The PUI exposes the various components and services offered by the environment. Through the PUI, users are able to define, update, and persist process models/programs during the modeling phase, instantiate a process model for enactment, participate in the enactment phase by performing manual and/or 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 PUI uses the *perspective* concept in a way similar to Eclipse's perspectives [26] to control the visibility and presentation of items in the workspace of the environment. The PUI includes three different perspectives to support the modeling, enactment, and management features of PCSE-KDD.

The Enactment Engine 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 during process modeling and enactment. The TIU manages the invocation of tools specified in the process program. Tools are specified in the resources section of the program and they can be referenced by the KDPMEL *action* construct. Two types of tools can be specified. The first type is interactive tools. The invocation of an interactive tool is based on a URL representing the tool executable. The second type is scripted tools that can be run from the KDPMEL *command* construct.

2.2 PCSE-KDD Perspectives

The PUI-Modeling perspective includes the items visible during the modeling phase along with their provided presentations and supported actions in the user interface. The PUI-Enactment includes the items relevant to the enactment phase. The PUI-Repository perspective includes the items maintained in the environment repository.

2.2.1 The PUI-Modeling Perspective

The PUI-Modeling perspective supports the three modeling approaches provided in KDPMEL: source-based, graph-based, and form-based. Fig. 3 sketches the layout of the PUI-Modeling perspective.

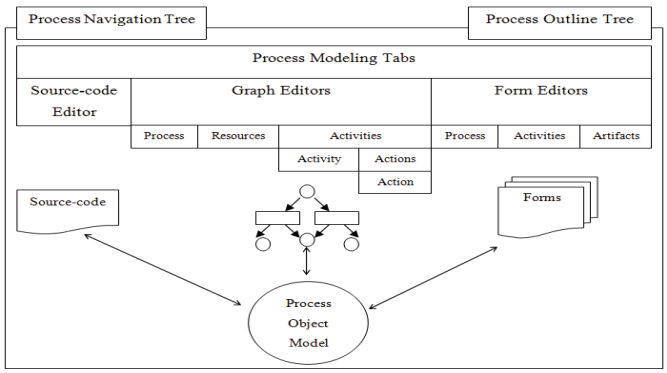

Fig. 3. The PUI-Modeling Perspective

The Process Navigation Tree allows for navigating between multiple processes. The Process Outline Tree displays the process components grouped by their types and allows for navigating between these components in the source-code representation of the process. The Process Modeling Tabs includes a tab for each process shown in the Process Navigation Tree. The Process tab includes a tab for the Source-code editor, a tab for the Graph editors, and a tab for the Form editors. The Graph Editors tab includes a tab for

the Process Graph editor, a tab for the Resources Graph editor, and a tab for the activities. The Activities tab includes a tab for each activity. The Activity tab includes a tab for the Activity Graph editor and a tab for the actions. The Actions tab includes a tab for each action. The Action tab includes the Action Graph editor. The Form Editors tab includes a tab for the Process Form editor, a tab for the activities, and a tab for the artifacts. The Activities tab includes a tab for each activity. The Artifacts tab includes a tab for each artifact. The Artifact tab includes the Artifact Form editor.

Fig. 4 depicts a view of the PUI-Modeling perspective showing its Source-code editor.

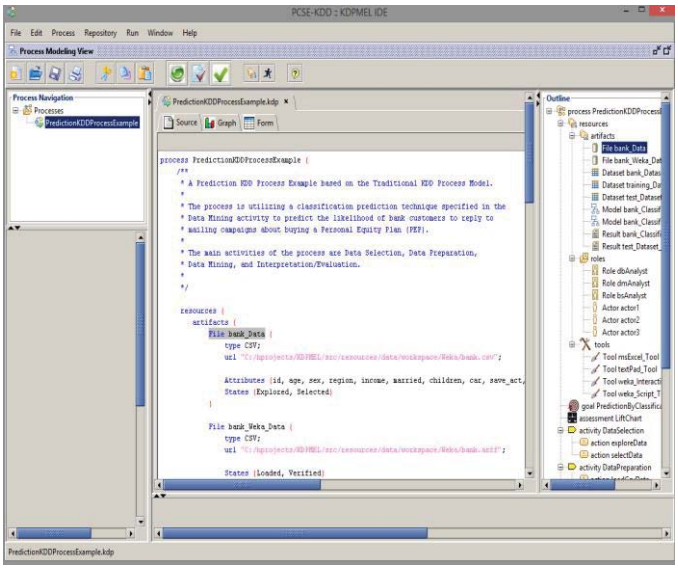

Fig. 4. The Source-code editor of the PUI-Modeling Perspective

A KDPMEL program can be developed using a Source-code editor and then updated using the Graph and Form editors that are automatically created from the program source-code [2]. The alternative is to use the various graph editors: the Process Graph editor to create the process and its activities; the Resources Graph editor for process resources; the Activity Graph editor for each activity along with its constituent actions; and the Action Graph editor for each action. Process graphs are translated into their source-code and form representations. Fig. 5 through Fig. 7 depict multiple views of the PUI-Modeling perspective showing some of its various Graph editors.

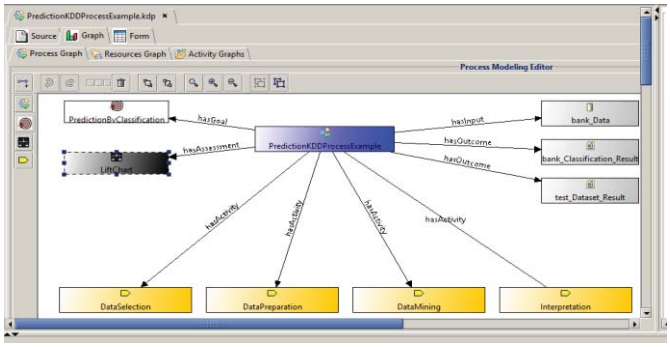

Fig. 5. The Process Graph editor of the PUI-Modeling Perspective

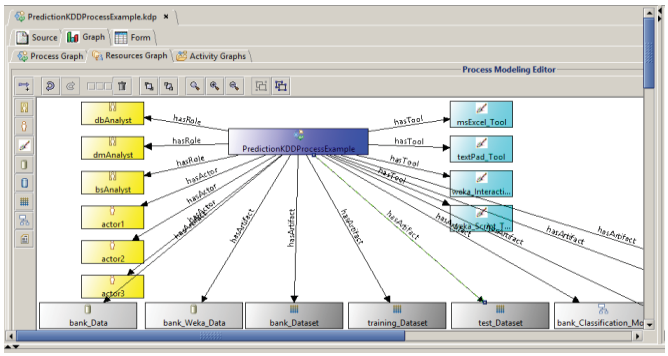

Fig. 6. The Resources Graph editor of the PUI-Modeling Perspective

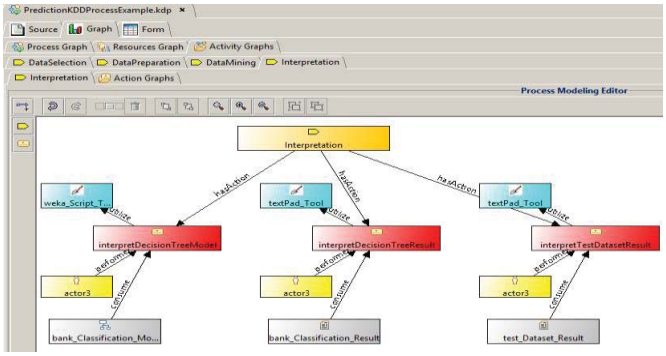

Fig. 7. The Activity Graph editor of the PUI-Modeling Perspective

2.2.2 The PUI-Repository Perspective

The PUI-Repository perspective manages the process resources using forms that are created to display and update the properties of these resources. In addition, a read-only graph is provided to show the flow of artifacts in the process starting from the process inputs to its outcomes. Fig. 8 sketches the layout of the PUI-Repository perspective.

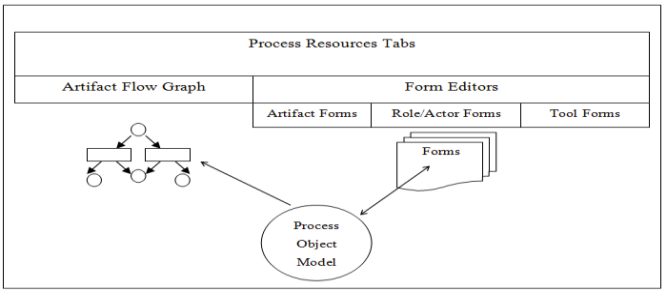

Fig. 8. The PUI-Repository Perspective

The Process Resources Tabs includes a tab for the Artifact Flow Graph and a tab for the Form editors. The Form Editors tab includes a tab for the artifact forms, a tab for the role/actor forms, and a tab for the tool forms. The Artifact Forms includes a tab for each artifact. The Role/Actor Forms includes a tab for each role/actor. The Tool Forms includes a tab for each tool. The Artifact tab includes the Artifact Form editor. The Role/Actor tab includes the Role/Actor Form editor. The Tool tab includes the Tool Form editor. Fig. 9 depicts a view of the PUI-Repository perspective showing its Artifact Flow Graph.

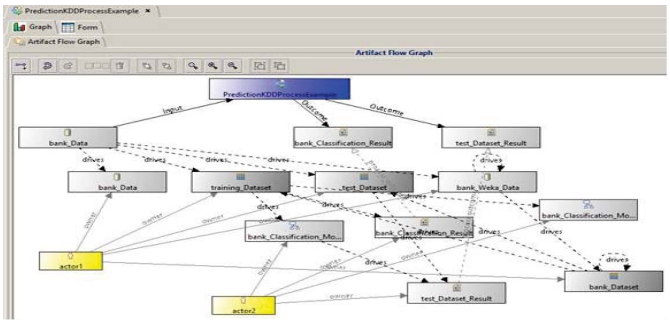

Fig. 9. The Artifact Flow Graph of the PUI-Repository Perspective

2.2.3 The PUI-Enactment Perspective

The PUI-Enactment perspective includes the items visible during the enactment phase along with their provided presentations and supported actions in the user interface. Fig. 10 sketches the layout of the PUI-Enactment perspective.

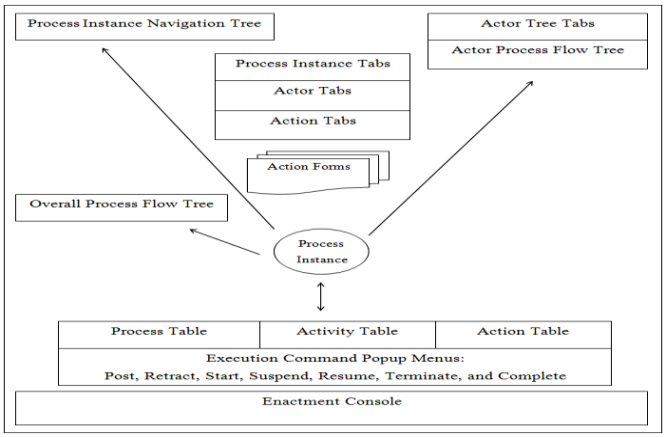

Fig. 10. The PUI-Enactment Perspective

The Process Instance Navigation Tree allows for navigating between multiple process instances for the same process or for different processes. The Overall Process Flow Tree and Actor Process Flow Tree represent process tasks as nodes that change their color based on the execution state of the task (Posted=Orange, [Started, Resumed]=Green, Suspended=Red, [Completed, Terminated]=Blue, Otherwise=Black). The Process Instance Tabs include a tab for each process instance. The Actor Tree Tabs includes a tab for each actor showing the Actor Process Flow Tree. The Actor Process Flow Tree groups and displays the tasks assigned for the actor. The Actor Tabs includes a form for each actor showing the actions assigned to the actor. The Action Forms display detailed action information. The Process Table displays process instances, the Activity Table displays activity instances, and the Action Table displays action instances. Each task in these tables is displayed with its execution state and performing actor name along with other execution information such as its running time. A popup menu is displayed showing applicable execution commands to select from. The Enactment Console displays execution information.

Fig. 11 depicts a view of the PUI-Enactment perspective.

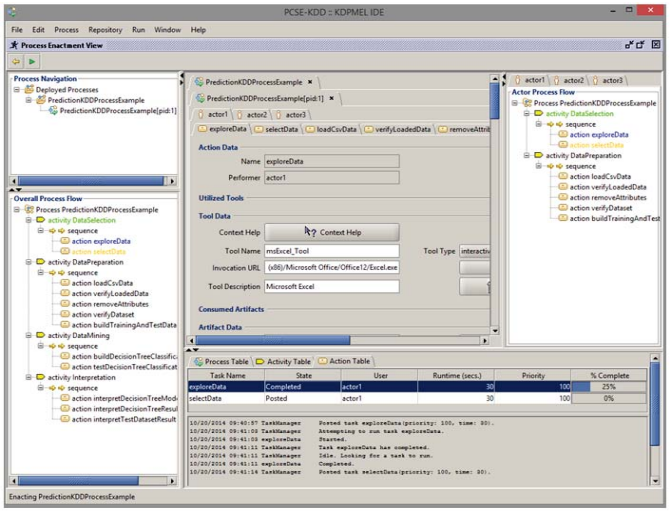

Fig. 11. Enacting a Process Instance in the PUI-Enactment Perspective

2.3 The Enactment Engine

The main components of the Enactment Engine are:

- The KDPMEL Interpreter
- The Repository Management Unit (RMU)
- The Tool Invocation Unit (TIU)

2.3.1 The KDPMEL Interpreter

The user interactions with KDPMEL Interpreter are performed through the PUI-Enactment perspective.

The execution of a process program starts by issuing an execute command, which causes the PUI-Enactment perspective to be created and presented (Fig. 11) to the user. A process instance is created, placed in a *posted* state, and entered in the process table (Fig. 12). The user is offered a menu of valid transition states; e.g., a *posted* task can be either *retracted* or *started.* The same mechanism applies for activities and actions that are ready for execution: they are placed in a *posted* state in the activity and action tables. Starting a process instance triggers the execution of its activities in the order they are defined. Starting an activity triggers the execution of its sub-activities in a depth-first order and its constituent actions based on their control construct.

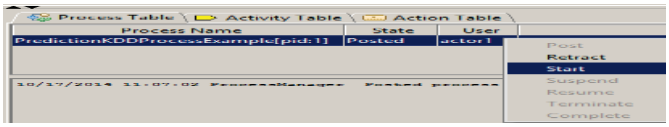

Fig. 12. A *Posted* KDPMEL Process Instance

The Interpreter interacts with the PUI-Enactment perspective to update its presentation based on the execution states and to accept the user selection of choices offered during the execution.

To illustrate the dynamics of executing a KDPMEL *action*, consider the following example for building a decision tree classification model using the WEKA [14] framework:

```
process ADecisionTreeProcess {
     resources {
          artifacts {
              Dataset sampleDataset …
              Model sampleDecisionTreeModel …
          }
          roles { Actor dmAnalyst …}
          tools { Tool weka_Script_Tool …}
     }
     …
     action buildDecisionTreeClassificationModel {
          consume sampleDataset;
          produce sampleDecisionTreeModel;
          performer dmAnalyst;
          utilize {
             call weka_Script_Tool {
              command buildDecisionTreeCommand {
                kind Modeling;
                input sampleDataset;
                output sampleDecisionTreeModel;
                operation
                     "weka.classifiers.trees.J48";
                parameters "-C 0.25 -M 2";
     …
}
```

The Interpreter establishes a handle on the action's consumed (*sampleDataset* artifact) and produced (*sampleDecisionTreeModel* artifact) artifacts by submitting queries to the RMU. Each artifact handle contains information, such as *name*, *type*, *URL*, etc., which allows accessing and controlling the artifacts. Another handle is established for each utilized tool (*weka_Script_Tool* tool). The Interpreter sends the TIU a tool handle that specifies the tool description and how to invoke it (invocation *URL*/*command*). Two mechanisms are used for tool invocations. Simple invocation is provided for an interactive tool through calling the tool's URL. A tool that can be called in a scripted mode is invoked through a plug-in module that implements the translation of KDPMEL external commands (*buildDecisionTreeCommand* command) into their appropriate commands that are accepted by the tool.

The interpreter identifies the assigned actor (*dmAnalyst* actor) by issuing a query to the RMU and accordingly notifies that actor. An action that awaits a user's response is in the *posted* state. The actor would respond through the PUI-Enactment perspective by selecting his/her preferred choice. When the actor starts the action, the interpreter identifies the needed artifacts, binds them with the action, identifies the tools utilized by the action, and issues their invocation calls. The actor then takes responsibility for executing and finishing the action and informing the interpreter through the PUI-Enactment perspective about the completion status. If the action is completed successfully, the interpreter issues an update or create request to the RMU for the produced artifacts and determines the next action to be executed.

When *buildDecisionTreeClassificationModel* action becomes ready for execution, it is placed in a *posted* state on the action table. The color of the node representing the action in the Overall Process Flow Tree and Actor Process Flow Tree changes to *Orange*. The actor performing the action starts its execution by clicking on the row representing the action in the action table and selecting the *Start* choice from the popup menu. A number of dialogs begin with the actor to support the performance of the action, the state of the action changes to *started* state, and the action node color changes to *Green*. The first dialog asks for invoking the *weka_Script_Tool* tool that is utilized by the action and the actor responds with *Yes*. The second dialog asks whether to run the commands associated with the tool and the actor responds with *Yes*. The third dialog asks to run the first tool command (*buildDecisionTreeCommand* command) and the actor responds with *Yes*. The last dialog confirms that the command has been executed successfully. Fig. 13 illustrates the execution of the action.

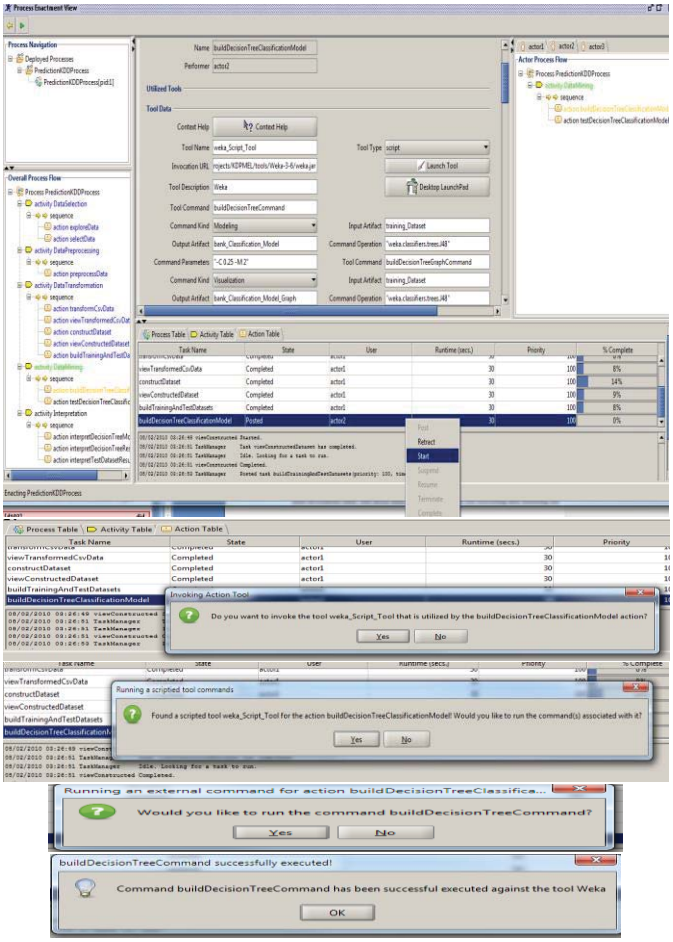

Fig. 13. The execution of the *buildDecisionTreeClassificationModel* action

2.3.2 The Repository Management Unit (RMU)

RMU provides management and persistence support for process resources. In addition, process instances that are created during enactment are maintained by the RMU.

Concurrent access to the process artifacts can happen in a number of situations and there is a need for a concurrency control mechanism, suitable for KDD processes, to handle these situations. KDD processes are highly interactive and iterative. They include tasks that can have very long running times and a high degree of interactions with process actors.

These KDD tasks are similar to database transactions that involve browsing or performing data entry, which may last several minutes. For this kind of transactions, concurrency can severely suffer with higher isolation levels. Also, the possibility of deadlock is increased. A lower isolation level is typically used for this kind of transactions.

Given the nature of KDD tasks and their running time similarities with long running database transactions, the lowest isolation level would be a better choice. The RMU supports the lowest isolation level as defined in the ANSI SQL Standard [6] by implementing a simple exclusive locking mechanism on the process artifacts that are updated by an action. Process artifacts that are updated by an action are locked for exclusive use (write-lock) when the action is started (the *started* state) and released when the action is finished with either *terminated* or *completed* state.

2.3.3 The Tool Invocation Unit (TIU)

TIU is responsible for orchestrating and managing the invocation of tools. Two mechanisms are used for tool invocations. Simple invocation is provided for an interactive tool through calling the tool's URL. A tool that can be called in a scripted mode is invoked through a plug-in module that implements the translation of KDPMEL external commands into their appropriate commands that are accepted by the tool. Fig. 14 illustrates the structure of the TIU.

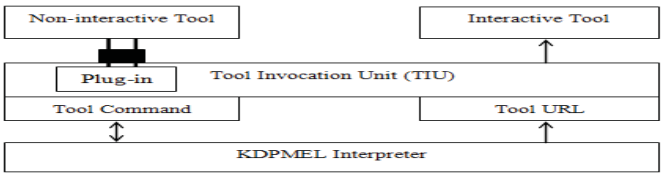

Fig. 14. The Tool Invocation Unit (TIU)

2.4 PCSE-KDD Key Aspects and Novelty

PCSE-KDD combines aspects form KDD Support Environments, Language Integrated Development Environments (IDEs), and PCSEEs. These aspects are blended together in PCSE-KDD to support developing KDD processes in a language-based and process-oriented approach.

As a KDD support environment, PCSE-KDD provides features to assist KDD developers, maintain KDD resources, invoke and interact with KDD supporting tools, automate routine KDD tasks, manage dependencies and interactions between KDD techniques, and enforce mandatory KDD rules and practices.

As an IDE for KDPMEL, PCSE-KDD provides basic features to facilitate the construction, execution, and management of KDPMEL programs. KDPMEL programs are constructed using a hybrid modeling approach, mixing different types of editors and views in source-based, graph-based, and form-based styles.

As a PCSEE, PCSE-KDD exploits an explicit representation of the KDD process (a process program) to support KDD development activities and to manage the overall process. PCSE-KDD implements the process definition/instantiation/enactment paradigm to develop KDD processes in a way similar to developing software processes in PCSEEs.

We believe that this novel approach for designing PCSE-KDD combines the benefits of the underlying aspects (KDD, Language, and Process) to provide concrete, flexible, explicit, and process-oriented support for KDD processes.

We have used PCSE-KDD to implement a non-trivial KDD prediction process in [1]. To evaluate PCSE-KDD, we have also implemented a real-world case study data analysis process for analyzing, comparing, and visualizing streams of ocean data [3]. The evaluation results of this case study show that the entire process can be fully automated, which yields an execution time of 4.8 hours as opposed to 79 hours original execution time for the manual non-process implementation. This is a 16.5x speedup over the original execution time. The results also show that the execution of the process can be easily repeated with the same or different configurations and/or data. Moreover, the results show that minor to moderate program adjustments and configurations are needed to expand and scale up the analysis. As for novice users, the results show that they will have no difficulty in executing the analysis in PCSE-KDD.

3 Prototyping PCSE-KDD

The environment has been prototyped in Java utilizing a number of open source libraries and tools such as *JavaCC* [5], *JGraph* [27], *JGoodies* [16], and *SMC* [15]. The environment has the look and feel of Eclipse IDE. It also has similar Workbench that includes three different perspectives for the Modeling, Enactment, and Management functionalities.

The prototype is structured into multiple modules: the KDPMEL language along with its components and tools (Parser, Model Checker, Interpreter, Source-code Editor, etc.); the Process Object Model (Process Components); the Desktop Development Environment (Workbench, Graph Editors, and Form Editors); the Runtime System (Enactment Engine and State Diagrams); and the Repository.

In addition to the PCSE-KDD Java prototype, an Eclipse Rich Client Platform (RCP) Plug-in version has been prototyped to provide full modeling and management capabilities that are supported by the Eclipse platform. With Eclipse RCP, a plethora of Eclipse features and components are available for reuse. Building on a platform facilitates faster development and seamless integration. The inherit extensibility of Eclipse allows to build not only a closed-form application, but also and open-ended platform like the Eclipse IDE itself. This RCP version utilizes the following technologies:

- The Eclipse Workbench, Perspectives, Views, and Editors for building the Desktop of the environment.

- The Eclipse Modeling Framework (EMF) [11] for defining the KDD Process Meta-Models.
- The Eclipse Graphical Modeling Framework (GMF) [12] and Graphical Editing Framework (GEF) [13] for building the KDD Process Graph Editors.
- The *xText* Language Development Framework [17] for developing KDPMEL.

4 Conclusions

In this paper, we presented the Process-Centered Support Environment for KDD (PCSE-KDD) that can be used to develop KDD processes in a way that is similar to developing software processes, which is based on encoding KDD processes as process programs written in KDPMEL and exploited by PCSE-KDD to provide execution support and management for KDD processes.

PCSE-KDD includes a number of modeling editors and views, an Enactment Engine for runtime process execution support, and a Repository for providing persistence support for the process resources. PCSE-KDD has been prototyped in Java plus a number of open source libraries and tools.

In PCSE-KDD, 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. Also, 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 employs an engineering approach to develop KDD processes. It is a language-based and process-driven approach. 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 expanding the support for more KDD tools and continuing the development of PCSE-KDD to provide more enhanced graphical modeling and management for KDD artifacts.

5 References

[1] Mansour, H. A., Duchamp, D., and Krapp, C.-A. *A Language-Based and Process-Oriented Approach for Supporting the Knowledge Discovery Processes*. In Proceedings of the 11th International Conference on Data Mining (DMIN'15) (pp. 107-115), July 2015.

[2] Mansour, H. A. (in press). *KDPMEL*: *A Knowledge Discovery Process Modeling and Enacting Language*. The 12th International Conference on Data Mining (DMIN'16), July 2016.

[3] Mansour, H. A. *A Process-Centered Environment for Modeling, Enacting, and Managing the Knowledge Discovery Processes*. PhD dissertation, Stevens Institute of Technology, Hoboken, N. J., 2015.

[4] David Jensen et al. *Coordinating Agent Activities in Knowledge Discovery Processes*, Department of Computer Science, University of Massachusetts Amherst, 1999.

[5] The *JavaCC* Framework. URL: https://javacc.dev.java.net/

[6] The SQL-92 Standard. URL: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt.

[7] L. Kurgan and P. Musilek. *A Survey of Knowledge Discovery and Data Mining Process Models*. Knowledge Engineering Review, 21(1), pp. 1-24, 2006.

[8] Marban, O. et al. *An Engineering Approach to Data Mining Projects. Intelligent Data Engineering and Automated Learning* – IDEAL 2007, LNCS 4881, pp. 578-588, 2007.

[9] Marban, O. et al. *Toward data mining engineering: A software engineering approach*. Information Systems 34 (1), 2009.

[10] Pohl, K. et al. *PRIME-Toward Process-Integrated Modeling Environments*. ACM Transactions on Software Engineering and Methodology, Vol. 8, No. 4, October 1999, Pages 343-410.

[11] The Eclipse Modeling Framework (EMF). URL: http://www.eclipse.org/modeling/emf/

[12] The Eclipse Graphical Modeling Framework (GMF). URL: http://wiki.eclipse.org/Graphical_Modeling_Framework/

[13] The Graphical Editing Framework (GEF). URL: http://www.eclipse.org/gef/

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

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

[16] The JGoodies Framework. URL: https://jgoodies.dev.java.net/

[17] The *xText* Language Development Framework. URL: http://www.eclipse.org/Xtext/

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

[19] Cinara Ghedini and Karin Becker. *A documentation model for the KDD application management support*. Faculdade de Informatica, PUCRS 2000.

[20] Osterweil, L.J. *Software Processes are Software Too*. In Proceedings of the Ninth International Conference on Software Engineering, pp 2-14, 1987.

[21] B.C. Warboys et al. *Collaboration and Composition: Issues for a Second Generation Process Language*. 1999.

[22] Vincenzo Ambriola et al. *Assessing Process-centered Software Engineering Environments*. Universita di Pisa, NTH-Trondheim, Politecnico di Milano, 1996.

[23] 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.

[24] 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, August 27, 2003.

[25] J. Segovia. *Definition and Instantiation of an Integrated Data Mining Process*. Jornadas de Seguimiento de Proyectos, 2007.

[26] Using Perspectives in the Eclipse UI. URL: https://www.eclipse.org/articles/using-perspectives/PerspectiveArticle.html

[27] The JGraph Framework. URL: http://www.jgraph.com/jgraph.html