NOTES ON THE ARCHITECTURAL DESIGN OF TMINER
Design and Use of a Component-based Data Mining Framework

Fernando Berzal, Juan-Carlos Cubero and Aída Jiménez
Department of Computer Science and Artificial Intelligence, University of Granada
ETSIIT, C/ Periodista Daniel Saucedo Aranda, s/n, 18071 Granada, Spain

Keywords: Data Mining, component-based systems, application frameworks, software architecture, design patterns.

Abstract: This paper describes the rationale behind some of the key design decisions that guided the development of the TMiner component-based data mining framework. TMiner is a flexible framework that can be used as a stand-alone tool or integrated into larger Business Intelligence (BI) solutions. TMiner is a general-purpose component-based system designed to support the whole KDD process into a single framework and thus facilitate the implementation of complex data mining scenarios.

1 INTRODUCTION

Traditional on-line transaction processing systems, also known as OLTP systems, work with relatively small chunks of data at a time, while on-line analytical processing systems, or OLAP systems, require the analysis of huge amounts of data (Chaudhuri and Dayal, 1997). It comes as no surprise that OLAP systems have very specific needs that conventional application frameworks do not properly address. This fact has led to the development of data mining (Tan et al., 2006) (Han and Kamber, 2006) and data warehousing (Widom, 1995) (Kimball and Ross, 2002), which try to satisfy the expectations of the so-called knowledge workers (executives, managers, and analysts).

This paper describes the rationale behind some key design decisions that led to the development of a component-based data mining framework called TMiner. As we will see, TMiner can be used as a flexible stand-alone data mining tool, but it has also been designed so that it can be easily incorporated into larger Business Intelligence solutions.

It should be noted that the tools and techniques TMiner collects somewhat overlap with existing Machine Learning algorithm collections, such as Weka (Witten and Frank, 2005). However, TMiner is more that a mere collection of independent algorithms for data mining tasks that can be directly applied on prepared datasets or invoked from your own code.

Some open-source and commercial data mining libraries (Prudsys, 2008) (Rapid-I, 2008) include facilities for their integration into actual enterprise systems. TMiner also provides usage modes specially designed for its tight integration into larger solutions.

TMiner is a general-purpose component-based system designed to support the whole KDD process into a single framework and thus facilitate the implementation of complex data mining scenarios. In this sense, TMiner is designed to be useful in a wide variety of application domains, in sharp contrast to domain-specific data mining systems such as iKDD or SA. While the Interactive Knowledge Discovery and Data mining system, iKDD, was designed for particular bioinformatics-related problems (Etienne et al., 2006), Perttu Laurinen's Smart Archive, SA, has been proposed for implementing data mining applications using data streams. (Laurinen et al., 2005)

The rest of our paper is organized as follows. Section 2 describes the architectural design of the TMiner framework and its component model. Section 3 describes the facilities TMiner offers for different usage scenarios, from the casual user who wants to perform simple data analysis tasks and the researcher who needs a more thorough experimentation, to the systems integrator who needs to incorporate data mining features into final solutions. Finally, Section 4 concludes our paper with some comments on the current status of TMiner and our expectations for its future.

Berzal F., Cubero J. and Jiménez A. (2008).
NOTES ON THE ARCHITECTURAL DESIGN OF TMINER - Design and Use of a Component-based Data Mining Framework.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 98-103
DOI: 10.5220/0001870600980103

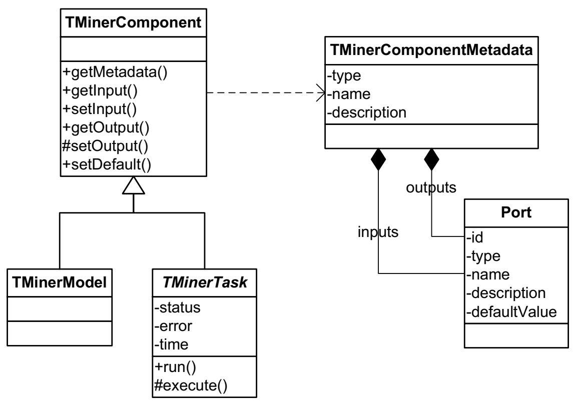

Figure 1: TMiner component model base classes.

2 TMINER COMPONENT MODEL

This section describes the TMiner component model, what TMiner components look like and the basic infrastructure provided by the TMiner framework for the design and use of new components.

2.1 TMiner Components

A software component is "a unit of composition with contractually defined interfaces and explicit context dependencies" (Szyperski et al., 2002). These context dependencies are specified by stating the required interfaces and the acceptable execution platform(s) for the software component. TMiner components have well-defined I/O ports whose specific properties can be easily consulted by means of the `TMinerComponentMetadata` object that is attached to every `TMinerComponent` (see Figure 1).

Software components are units of independent deployment, in contrast to objects in object-oriented programming (OOP), which are mere units of instantiation. Components are also units of third-party composition and they typically have no (externally) observable state. However, as objects in OOP, they encapsulate their state and behavior.

Obviously, components still act through objects in object-oriented systems, but they do not only contain classes. They can also contain additional resources. In TMiner, components must include a component descriptor describing its I/O ports and any additional metadata that might be required for the component to be used in practice. Figure 2 shows an example of such a descriptor in XML format. The component descriptor includes the names, textual descriptions, and required interfaces for all the component

```
<?xml version="1.0" encoding="utf-8" ?> <component>

 <type>tminer.kdd.association.ItemsetMiner</type>
 <name>Itemset miner</name>
 <description>TMiner base itemset miner</description>

 <inputs>
  <input>
   <id>dataset</id>
   <name language="en">Dataset</name>
   <description language="en">Input dataset</description>
   <type>tminer.data.Dataset</type>
  </input>
  <input>
   <id>encoder</id>
   <name language="en">Dataset encoder</name>
   <description language="en">Dataset encoder</description>
   <type>tminer.data.instance.DatasetEncoder</type>
  </input>
 </inputs>

 <outputs>
  <output>
   <id>itemsets</id>
   <name language="en">Itemset collection</name>
   <description language="en">Itemsets</description>
   <type>tminer.kdd.association.ItemsetCollection</type>
  </output>
 </outputs>

</component>
```

Figure 2: TMiner component descriptor.

I/O ports, as well as default values for component parameters. This information is extremely useful, for instance, in the automatic generation of the user interface for data mining tools (component developers do not need to worry about user interface issues and they can just focus on the development of the component themselves).

In a data mining framework such as TMiner, the user has to analyze large datasets with the help of mining tools and techniques. Data is gathered from different data sources and data mining algorithms are used in order to build knowledge models (Berzal et al., 2002). Hence TMiner components fall into two main categories:

- `TMinerModels` represent the entities data miners work with. They may provide the information our user needs to access different data sources (i.e. dataset metadata). They can also be descriptive or predictive models built from those data sources. They can even be used as the input to other mining algorithms in order to solve second-order data mining problems.
- `TMinerTasks` represent the tasks data miners must perform to analyze data. They are the active

objects that users need to build new models.

All components in TMiner must be serializable (i.e. they can be stored as byte sequences for later reconstruction). This is necessary because `TMinerModels` must be stored for later use, while `TMinerTasks` might need to be transferred to different processing nodes in a distributed computing environment.

Additionally, TMiner core classes include some standardized interfaces designed to simplify the representation, distribution, and use of the models discovered in the KDD process. For instance, `XMLable` components can be represented as XML documents, hence facilitating their storage for later use, as well as their visualization in standard web browsers (with the help of the corresponding XSLT style sheets). Likewise `SQLable` components, such as many different kinds of symbolic classification models, can be converted into SQL scripts that might be invaluable in practice, since they provide a very convenient method for using such models on relational databases.

Any information system can be described by a structure, a mechanism, and a policy following Perry and Kaiser's SMP model (Perry and Kaiser, 1991). In our case, TMiner models determine the structure of the system. TMiner tasks, which are responsible for the implementation of data mining techniques, are the mechanisms that let us solve data mining problems. Finally, the set of rules and strategies imposed by the system environment are used to establish its usage and security policies. This is the job of the TMiner framework we now proceed to describe.

2.2 The TMiner Framework

A component framework is "a collection of rules and interfaces (contracts) that govern the interaction of components plugged into the framework" (Szyperski et al., 2002). Component frameworks can also be seen as components that plug into higher-level component frameworks (e.g. when integrated into larger solutions).

Component-based frameworks are intended to help developers to build increasingly complex systems, enhance productivity and promote component reuse by means of well-defined patterns (Fayad and Schmidt, 1997) (Larsen, 2000). Such frameworks are widely-used in enterprise information systems, but they usually only provide low-level information processing capabilities, since they are OLTP-application-oriented. In contrast, TMiner is a component-based framework that has been custom-tailored to solve decision support problems, even though it should be noted that its approach could also be of use in a wide range of scientific computing applications.

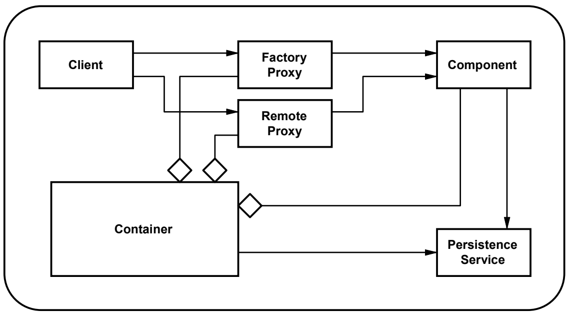

Figure 3: The Enterprise Component Framework.

Most commercial component-based frameworks, such as Sun Microsystems Enterprise JavaBeans or the Microsoft .NET Common Language Runtime, are based on a common architectural pattern, known as the Enterprise Component Framework (Kobryn, 2000). A simplified representation of this pattern is depicted in Figure 3. This pattern contains six roles, shown as rectangles in the figure, whose responsibilities can be described as follows:

- **Clients** are the entities that request services from a component in the framework. End users, component developers, researchers, and other automated systems may act as clients in TMiner (see Section 3 for more information on TMiner usage modes).
- **Components** provide the services requested by clients. As mentioned above, both data access modules and knowledge models are full-fledged components in TMiner. Data mining algorithms could also be considered as components on their own, but they are just used through factory proxies to build knowledge models.
- **Proxies** relay calls from clients to components. This level of indirection is hidden from the client perspective and it makes location transparency possible (when needed, it also supports message interception). *Factory proxies* perform object factory operations that are common to all framework components (such as creation or retrieval), while *remote proxies* handle operations that are component-specific (e.g. inspection and parameter setting). Proxies are usually supported by the reflection capabilities found in modern computing platforms.
- The **Container** represents the framework's runtime environment. The container provides distributed computing services, load balancing, interprocess communication, security, persistence, resource discovery, and hot deployment mech-

anisms. Transactions are usually supported by enterprise frameworks but are not needed in the TMiner data mining framework. TMiner, however, needs specific scheduling, monitoring, and notification mechanisms to manage data mining tasks.

- Finally, the **persistence service**, typically managed and coordinated by the container, is used for the storage and retrieval of framework components.

Component-based frameworks are suitable for large-scale systems because they provide a solid foundation upon which whole applications and product lines can be deployed in a systematic and controlled way. The TMiner architecture makes the development of new techniques and algorithms faster. Once a new component has been devised and tested in the laboratory, it can be easily deployed into production environments with the help of a simple component descriptor. This simplifies maintenance and upgrade efforts, hence smoothing the evolution any system must face during its lifetime.

2.3 TMiner Subsystems

Apart from the common infrastructure needed to build typical data mining tools, TMiner provides support for flexible data access and many of the most common data mining algorithms and techniques:

- The **TMiner data access subsystem** acts as its extract, transform, and load (ETL) front-end and it makes heavy use of well-known object-oriented design patterns for improving its flexibility (Gamma et al., 1995).
- The **TMiner knowledge-discovery subsystem** provides a wealth of classification models and clustering techniques, as well as efficient association rule mining components and anomaly detection tools.

3 TMINER USAGE MODES

System usability is critical for user acceptance. Provided that knowledge workers are not necessarily knowledgeable about computers, TMiner must provide different usage modes for different usage scenarios.

TMiner supports the progressive usage model or triphasic model. This model recognizes that patterns of usage evolve as users build experience and that supporting these patterns requires specific and somewhat different facilities within the user interface architecture (Constantine and Lockwood, 1999). TMiner, therefore, provides different usage modes for novice, intermediate, and advanced users:

3.1 Basic Usage Modes

Novice users tend to perform basic tasks, with a limited use of alternatives. The underlying complexity of the system should be invisible to this kind of users. TMiner acknowledges this fact and the default values typically included in a component descriptor let users employ a data mining component ‘out of the box’.

Beginners usually interact with the system by trial and error, and they have a strong dependence on help and guidance. For them, TMiner provides an attractive Web interface that incorporates novel visualization techniques in order to motivate exploration, such as VisAR (Techapichetvanich and Datta, 2005). This rule visualization technique is based on parallel coordinates, a common way of visualizing high-dimensional geometry and analyzing multivariate data (Inselberg, 1985)

The ability to share data among system users is another aspect that is closely related to a data mining system usability. TMiner lets casual system users to browse through already-discovered models and share their own models with other system users. This computer-supported cooperative work (CSCW) focus is especially relevant in data mining applications, where the discovered knowledge must be properly represented and communicated. (Berzal et al., 2003)

3.2 Intermediate Usage Modes

Intermediate TMiner users have expanding needs and they typically exhibit changing patterns of interactions. “Intermediates (those who are neither beginners nor old hands, and who make up most of the user population) are perhaps the most neglected user segment in terms of interface design, yet there are possibly more intermediate users than beginners or experts.” (Constantine, 1994)

TMiner component model facilitates the construction of data mining tools that let knowledgeable users tune data mining techniques by playing with the knobs TMiner components provide, i.e. their I/O ports. Component descriptors are extremely useful here, since they can be used to automatically generate the user interface needed for such knob-turning.

```
// Database connection details...
var database = {
      driver: "interbase.interclient.Driver",
      url: "jdbc:interbase://localhost/datasets.gdb",
      user: "SYSDBA",
      password: "*****" };

// Dataset...
var id = "CENSUS";
var dataset = new tminer.data.wrapper.jdbc.JDBCDataset
                    (id, database.user, database.password,
                         database.url,  database.driver);
// Attributes
var discreteAttributes   = dataset.getAttributeIDs
                              (tminer.data.type.StringType);
var continuousAttributes = dataset.getAttributeIDs
                              (tminer.data.type.NumericType);
```

Figure 4: TMinerScript code snippet needed to access a particular dataset through the standard JDBC API.

3.3 Advanced Usage Modes

Expert users' primary concerns are efficiency and productivity. They need to perform complex, sophisticated tasks that are often nonstandard or might be unsupported. They need interfaces that operate in multiple modes, frequently changed to fit the particular demands of the task at hand.

These advanced users can directly use TMiner components from their own *Java* code. They can even customize them by attaching dynamic ports to TMiner components without having to modify their source code nor create component subclasses. This can be helpful when monitoring system performance or addressing other cross-cutting concerns that appear in practice. *AspectJ*, an aspect-oriented extension for the Java programming language, can also be used with TMiner components to simplify the implementation of such cross-cutting concerns.

Sometimes, advanced users prefer faster methods to interact with TMiner components, without having to go through a complete edit-compile-build-run cycle each time they want to tweak anything. *TMinerScript* is included in TMiner for such users. TMinerScript is a scripting language that can be used to control TMiner using the syntax of JavaScript (Flanagan, 2006). Scripting languages such as TMinerScript, by being accessible to the end user, enable the behavior of an application to be adapted to the user's specific needs and thus provide the greatest possible degree of control to the user. Figures 4 through 6 show some simple TMinerScript scripts that can be used to perform common data mining tasks in TMiner.

```
...
// Classification model
var classifier = new classification.tdidt.TDIDTClassifier();
classifier.setDefault();

// Classifier parameters (common to all algorithms)
classifier.dataset = dataset; // e.g. JDBC Dataset
classifier.encoder = encoder; // ... to encode input data
classifier.classAttribute = "classLabel";
classifier.discreteAttributes = discreteAttributes;
classifier.continuousAttributes = continuousAttributes;

// Specific parameters (TDIDT algorithm)
classifier.divisionRule = "GainRatio";
classifier.pruning = true;
classifier.pruningCF = 0.25;
classifier.binarySplits = true;
classifier.balanceThreshold = 0.2;

// Classifier construction
classifier.build();

// Output (e.g. in SQL format)
classifier.toSQL("TABLE", "CLASS");
```

Figure 5: TMinerScript code snippet needed to build a decision tree classifier.

4 CURRENT STATUS AND FUTURE DIRECTIONS

We have described some of the main features of TMiner, a component-based data mining framework.

TMiner can be used as a stand-alone web-based data mining tool, providing components for many of the tasks we might need to analyze data, ranging from data access to knowledge discovery. Its current version lets users build classification models, cluster data, mine associations, and detect anomalies.

TMiner component model is also suitable for its integration into larger solutions whose requirements include some of the data mining features TMiner provides. In fact, TMiner offers alternative usage modes intended to facilitate such integration. On the one hand, it can be directly called from third-party code as any other component library. On the other hand, clients can use the scripting facilities TMiner provides for automating the execution of data mining tasks.

Our current research efforts focus on the development of novel data mining techniques (e.g. dealing with different kinds of data sources) as well as on the improvement of current data mining solutions by providing a scalable data mining system for scientific and business applications.

```
...
// Algorithm details
var algorithm
      = "tminer.kdd.classification.tdidt.TDIDTClassifier";
var parameters = new tminer.model.adt.Dictionary();

parameters.add("divisionRule", "GainRatio");
parameters.add("binarySplits", true); parameters.add("pruning",
true); parameters.add("pruningCF",    0.25);

// Cross-validation experiment
var experiment = new classification.CrossValidation();

experiment.type = algorithm;
experiment.parameters = parameters;
experiment.partitions = 10;

experiment.dataset = dataset;
experiment.encoder = encoder;
experiment.classAttribute = "classLabel";
experiment.discrete = discreteAttributes;
experiment.continuous = continuousAttributes;

experiment.run();

// Experiment results
experiment;
```

Figure 6: TMinerScript code snippet needed to run a cross-validation experiment.

ACKNOWLEDGEMENTS

Work partially supported by research project TIN2006-07262.

REFERENCES

Berzal, F., Blanco, I., Cubero, J. C., and Marín, N. (2002). Component-based data mining frameworks. *Communications of the ACM*, 45(12):97–100.

Berzal, F., Cubero, J. C., Marín, N., Serrano, J.-M., and Blanco, I. (2003). Usability issues in data mining systems. In *ICEIS 2003: Proceedings of the 5th International Conference on Enterprise Information Systems (Volume II - Artificial Intelligence and Decision Support Systems)*, pages 418–421.

Chaudhuri, S. and Dayal, U. (1997). An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74.

Constantine, L. L. (1994). Interfaces for intermediates. *IEEE Software*, 11(4):96–99.

Constantine, L. L. and Lockwood, L. A. D. (1999). *Software for Use: A practical guide to the models and methods of usage-centered design*. ACM Press / Addison-Wesley.

Etienne, J., Wachmann, B., and Zhang, L. (2006). A component-based framework for knowledge discovery in bioinformatics. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 916–921.

Fayad, M. E. and Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38.

Flanagan, D. (2006). *JavaScript: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Inselberg, A. (1985). The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91.

Kimball, R. and Ross, M. (2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc.

Kobryn, C. (2000). Modeling components and frameworks with UML. *Communications of the ACM*, 43(10):31–38.

Larsen, G. (2000). Component-based enterprise frameworks. *Communications of the ACM*, 43(10):24–26.

Laurinen, P., Tuovinen, L., and Roning, J. (2005). Smart archive: A component-based data mining application framework. In *ISDA'05: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 20–25.

Perry, D. E. and Kaiser, G. E. (1991). Models of software development environments. *IEEE Transactions on Software Engineering*, 17(3):283–295.

Prudsys (2008). XELOPES library - eXtEnded Library fOr Prudsys Embedded Solutions. http://www.prudsys.com/.

Rapid-I (2008). RapidMiner (formerly YALE, Yet Another Learning Environment). http://rapid-i.com/.

Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley.

Tan, P.-N., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Addison-Wesley.

Techapichetvanich, K. and Datta, A. (2005). VisAR: A new technique for visualizing mined association rules. In *ADMA 2005: 1st International Conference on Advanced Data Mining and Applications, LNCS 3584*, pages 88–95.

Widom, J. (1995). Research problems in data warehousing. In *CIKM '95, Proceedings of the 1995 International Conference on Information and Knowledge Management, November 28 - December 2, 1995, Baltimore, Maryland, USA*, pages 25–30. ACM.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.