Object Mining:
A Practical Application of Data Mining for the Construction and Maintenance of Software Components

Anders Torvill Bjorvand
Department of Informatics, University of Oslo.
P.O. Box 1080 Blindern, 0360 Oslo
Email: torvill@trolldata.no

Abstract. This paper adresses the issue of automatic construction and maintenance of software components by utilizing data mining techniques. Some methods are shown, and more specifically, an implementation on an IBM PC is demonstrated. This approach is particularly useful for the synthesis of embedded decision support components and intelligent agents.

1 Introduction

In the following, a working knowledge of the object-oriented paradigm and data mining is recommended.

The object-oriented paradigm, invented at the Norwegian Computing Center in the 1960s ([5]), has gained a tremendous momentum in both academia and industry, and is now widely regarded as the most important current methodology for the construction of large information systems.

Compared to previous approaches, these information systems are easier to both build and maintain since they are divided into autonomous parts; making delegation of responsibilities easier. A lower degree of coupling between different parts of the system is a natural consequence of this approach.

This paper introduces the notion of Object Mining which is the process/idea of using data mining for both creation of and maintenance of such components. This concept was first published in [4] . The current paper, however, emphasizes the practical aspects of using this technology. The example is based on a smaller example in [4] . It will become clear that this approach merges creation and maintenance in a way that gives full support to the entire life cycle of the system. Most practical investigations show that maintenance of a software system represents a greater cost than the initial creation.

1.1 Structure of this Document

I will explain how the Object Mining technology merges these techniques to become a powerful tool for both development and maintenance of computer software. Much of this technology is explained through an example from real life showing the entire process from data analysis to the finalized component.

First we will introduce Object Mining, then we will illustrate it with some examples, then we will describe an implementation of it.

1.2 Related Research

The idea of synthesizing software into components is new, but its ancestors are many.

The most important are the fields of embedded and distributed expert systems and intelligent agents. A fair overview of this can be seen in [9] .

The Object Mining technology is, as will become evident, related to the field of expert systems. Traditional expert systems were constructed as monolithic applications that corresponded with the world through a human being, but the adaptation to an embedded expert system seems a natural one, where the expert system automatically interacts with the rest of the system. This, however, would require changes to the intended object model of the system with respect to communications, interpreters, etc. The Object Mining technology produces components that fit easily into the existing object model. This is crucial to maintaining an elegant, understandable and therefore maintainable design.

2 Object Mining

An object can be viewed as a set of attributes/variables and a set of methods to set/get and/or analyze those attributes. For the great class of objects where this definition of an object holds true in a strict fashion, our method is applicable (some known exceptions are typical user interface objects and objects with attribute-values that are hard to discretize (typically mathematical functions and computational objects)). Data mining techniques take sets of attributes and their combinations of values as input. The product is a set of rules. By putting these attributes and rules together, we have an automatically produced object (also called software component in more modern terminology).

The Object Mining technology is the idea of utilizing data mining techniques to produce software objects. The choice of data mining techniques will vary with the domain that is being modelled, and is not part of the Object Mining technology.

To formalize, the Object Mining technology is best applied through the following five steps:

- Given an object model of your system, check which objects you either have or can easily sample data/measurements about.
- Given the databases for these objects, choose the parameters for which you want the component to supply you with results. These parameters are called the decision parameters. The set of decision parameters can very well be the entire set of parameters. This will often be the case in components intended to perform simulations.
- Apply data mining techniques to produce a set of rules on the form "if *param_1=6* and *param_2=34* and .... and *param_n = 56* then decision_param_1=4 and ... and decision_param_m=75". The initial number of parameters should also decrease because the dependency analysis performed by the data mining system should prove some parameters redundant/superfluous. This is provided for very nicely in many data mining techniques - a common one is the reducts of rough set theory.
- Create a component with the following features:
  - Internal variables representing all the parameters and decision parameters.
  - A data structure for the ruleset.
  - Methods for setting and reading the values of each variable, eg set_param_1(), get_param_1(), etc.
  - Methods retrieving the values of the decision parameters. These methods rely on data mining strategies for applying the ruleset.

- Incorporate these objects in your system according to the object model - interfaces might be an issue here.

The component described would be relatively simple. To use it, you set its conditional values and every time you want a decision value you ask for it explicitly - upon which it is calculated and returned. These tasks of interacting with the object might of course be delegated to several objects. Some objects might provide with data acquisition, some want to read the decisions etc.

A natural extension is an "active component" where the objects that rely on information from the component can be automatically informed of changes. This is accomplished by letting the dependent objects register with the component. Every time one of the parameters of the component changes, the set of decision parameters are immediately recalculated (without an explicit method invocation). If the values of one or more of the decision values have changed, an event informing of the change is automatically posted to the objects that have registered themselves.

Active objects fit well into the framework of process control and other systems where the object plays an active part. An obvious application domain where this sort of functionality would be required, is the field of intelligent agents. A more advanced way of registering for notification might be subscription lists with detailed information when each object should be notified. An object in charge of crisis situations might for instance only be noticed when the parameters reach a critical level.

3 Examples

We have a simple example concerning dogs. This example was introduced in [4] , but is elaborated on here.

3.1 Dog Recognizer Model

Figure 1 shows the UML () sequence diagram for a simple system for visual recognition of breeds of dogs. The system consists of a camera, a feature recognizer and the Object Mining generated component Dogs. The camera component supplies the feature recognizer with a picture of a dog, the feature recognizer extracts the characteristics of the dog's tail and hair and initializes the Dogs component with it. The Dogs component is then capable of informing the system of the breed of the dog. A more realistic example would of course have to handle exceptions and ambiguities.

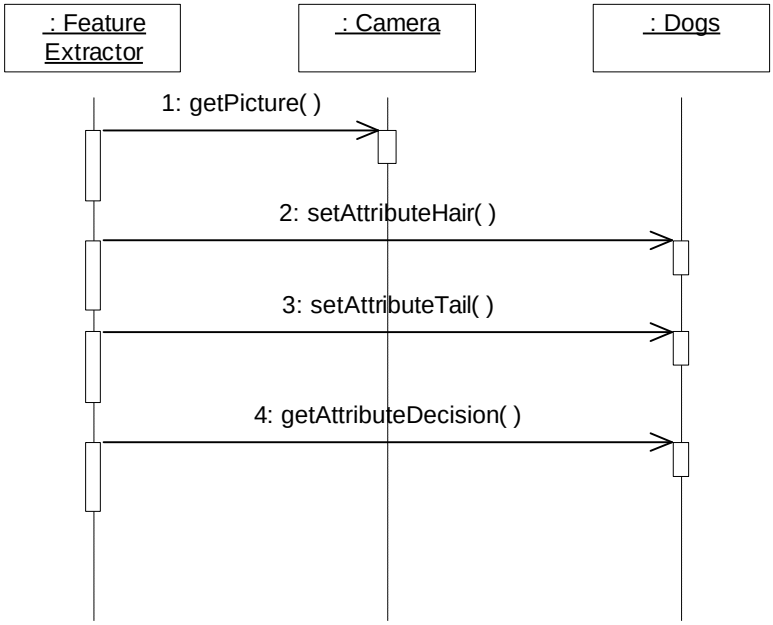

Figure 1 UML Sequence diagram for the dog recognition process

3.2 Development of the Dogs Component

We have a database of 10000 dogs. Each dog is stored in a record with several parameters like hair length, color, length of tail, breed etc. We have an image processing system that automatically extracts/recognizes all of these features except breed. We want a component that in each case can classify a dog based on certain features. Following the process outlined above, we get the following:

- We have a dog database, we choose the breed parameter as our only decision parameter.
- We apply data mining techniques and obtain the ruleset for determining the breed of dogs based on the information/experience in the database. Let's oversimplify and say that in our case, all other parameters than size and color are proved redundant in order to decide the breed. We get a ruleset with rules of the form: "if color=black and size=big then breed=Doberman Pincher"
- We can then automatically create a component with the following features:
  - Variables: color, size and breed.
  - A data structure for the rules.
  - Methods setting and reading the values of each variable:
    - set_param_size(), get_param_size()
    - set_param_color(), get_param_color()
    - The method: get_decision_breed() relying on data mining strategies for applying the ruleset.

To summarize, the Object Mining technology applies the techniques of data mining to the task/problem of producing objects/components.

4 Implementation

The crucial parts of the technology have already been implemented as an add-on to the data mining system "Rough Enough" which is also copyrighted by the author of this paper. "Rough Enough" as a pure data mining tool (without the Object Mining capabilities) was developed in 1995 at the Nowegian University of Science and Technology in Trondheim, Norway. The Object Mining capabilities have been added during 1997 and parts of 1998 and are part of version 4.0 of the Rough Enough system. Rough Enough is a general purpose data mining tool based on rough set techniques. It is further described in [1] , [2] and [3] .
The system can be downloaded from the internet address http://www.trolldata.com/.

4.1 Component Creation

The system will produce components conforming to the JavaBean standard from Sun Microsystems which is the most popular component standard right now built upon the Java language ([8]). OMG[1]s CORBA[2] IDL[3] files will also be automatically produced to allow the components more easily to be a part of an open distributed system.

The Java event model also supports the listener registration interface so that objects can register with the components developed through Object Mining so as to receive updates through the event system. Active objects, as explained in section 2, are therefore straightforward in Java.

4.2 The Dog Example Revisited

We will now return to the dog example and show a practical case of applying these methods.

Case

We have a table of information about dog breeds (Table 1). We want a component that can classify between these breeds, which makes the breed-attribute our decision attribute.

[1] OMG - Object Management Group - a consortium of leading information technology companies creating and maintaining industrial standards for object oriented development
[2] CORBA - Common Object Request Broker Architecture - an OMG standard for distributed access to objects
[3] IDL - Interface Definition Language - a middle language between the CORBA server and the different objects which may be implemented in Java, C++, Smalltalk, Cobol etc.

Table 1 . Dog Information System

Object number	Breed	Tail	Hair	Size
1	Doberman Pincher	Short	Short	Big
2	Dalmatian	Long	Short	Big
3	Cocker Spaniel	Short	Long	Small
4	German Shepherd	Long	Medium	Big

Processing with Data Mining Techniques

First, we import the data into a data mining system.

By applying data reduction methods, we get the result that tail and hair are sufficient with respect to deciding the breed. This makes the size parameter superfluous in our case.

By utilizing a method for synthesising rules, the ruleset is obtained. Each class of data mining techniques has its own methods of rules creation, so I will leave that alone for now.

Creation of a Component

A component is created from the rules. Currently, the system only supports JavaBeans conforming to the JDK 1.1 standard (the most recent Java standard at time of writing).

The system supports both passive and active objects and several rules-application strategies.

The particular component created in this example was a passive one called Dogs. It contains the following structure:

- Internal variables: AttributeTail, AttributeHair, AttributeDecision and Ruleset (double array)
- Methods:
  - Dogs(): constructor
  - getAttributeTail and setAttributeTail: sets and gets the tail attribute
  - getAttributeHair and setAttributeHair: sets and gets the hair attribute
  - calculateDecision: sets the value of the AttributeDecision variable utilizing the Ruleset variables
  - getAttributeDecision: invokes the method calculateDecision and then returns the value of the variable AttributeDecision

Listing of the Source for the Dog Component

The following code shows that each conditional attribute has an internal variable with external methods for setting and getting its value. The rules are represented with a double array, and the decision is calculated on demand. Due to space limitations, the full text of the method calculateDecision() could not be included. It is not part of the general framework either, so displaying it would not be very interesting - the method for applying rules should be chosen from the domain, possibly even on a situation to situation basis.

```
public class Dogs implements java.io.Serializable
{
   // Property declarations
   protected int AttributeDecision = 0;
   protected int AttributeTail = 0;
   protected int AttributeHair = 0;

   // Ruleset
   protected int[][] ruleSet = {
      {-1,  2,  4},
      {-1,  3,  3},
      {1,   1,  1},
      {1,   3,  3},
      {2,   1,  2},
      {2,   2,  4},
   };

   // Constructor
   public Dogs()
   {
   }

   protected void calculateDecision()
   {
      // here, different schemes for
      // rule-interpretation can be
      // implemented
   }

   public void setAttributeTail(int attribute_value)
   {
      AttributeTail = attribute_value;
   }

   public int getAttributeTail()
   {
      return AttributeTail;
   }

   public void setAttributeHair(int attribute_value)
   {
      AttributeHair = attribute_value;
   }

   public int getAttributeHair()
   {
      return AttributeHair;
   }

   public int getAttributeDecision()
   {
      calculateDecision();
      return AttributeDecision;
   }
}
```

Using the Component

Figure 2 shows a Java Applet utilizing the Dogs component. The Applet was created in the Symantec Visual Cafe Java environment. The Applet consists of textfields, a button, the Dogs component and a component that translates between the domain (tail-lengths, breeds) and our discretized component working internally in integers. The Dogs component was used as is, and did not need manual refinement. When the Tail and Hair parameters are given, pressing the Calculate-button gives the name of the Breed.

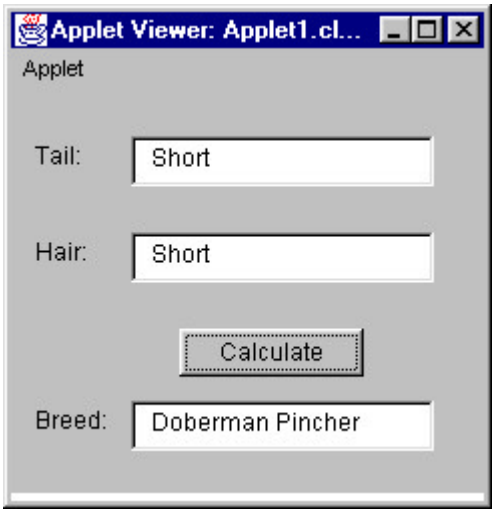

Figure 2 Java Applet using the Dogs JavaBean

5 Conclusion

We have identified the possibilities and importance of utilizing data mining techniques for synthesis of objects from experimental data. Several examples have been given, and an implementation is described.

The application of data mining to software engineering, called Object Mining in this paper, shows great promise, and will hopefully be further investigated in the future.

6 Future Research

The techniques outlined here suggests that a central system and a database might generate components based on the latest information with regular intervals. The next logical step would be to let these components enhance themselves through rule improvement by feedback. This opens up several important applications for embedded decision support systems and autonomous intelligent agents. A problem, especially with embedded systems, is that a lightweight object like the ones outlined in this paper often will be the only feasible way to go. A single superobject might sound glorious, but it might be too large or too inefficient.

By automating software construction relying on experimental data, the issue of correctness immediately becomes a major one. Completeness, inconsistency and handling of indeterminate decisions should be further investigated. There is a great potential for automating several of these tasks more easily than in traditional approaches to correctness preservation of programs.

Creating general "superobjects" with their own rule storage and techniques for exploring their own domain, a total system can be created with only one object class. In many ways, this is only a different perspective on agent oriented programming. I believe that an integration of that approach and the principles of genetic programming would be a fruitful direction of research.

Ways of integrating the Object Mining approach with modelling is an important field of study; especially within the new UML standard ([6] and [7]).

Acknowledgements

I would like to thank my wife Annette for her support and my little daughter Susanne for her cheerful smile. I would also like to thank the people at Oslo Research Park for fruitful discussions concerning the Object Mining technology.

References

[1] Bjorvand, Anders Torvill (1997). *'Rough Enough' - A System Supporting the Rough Sets Approach.* Proceedings of the Sixth Scandinavian Conference on Artificial Intelligence 1997. Helsinki, Finland. IOS Press, Amsterdam.

[2] Bjorvand, Anders Torvill and Komorowski, Jan (1997). Practical Applications of Genetic Algorithms for Efficient Reduct Computation. Proceedings of the 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics. Wissenschaft & Technik Verlag, Berlin.

[3] Bjorvand, Anders Torvill (1997). *Rough Enough - Software Demonstration*. Proceedings of the 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics. Wissenschaft & Technik Verlag, Berlin.

[4] Bjorvand, Anders Torvill (1998). *Synthesis of Objects: A Rough Sets Approach to Automatic Construction and Maintenance of Software Components*. European Conference on Artificial Intelligence, August, 1998. Workshop entitled Synthesis of Intelligent Agent Systems from Experimental Data.

[5] Dahl, O.J., Myrhaug, B. and Nygaard, K. (1968). *SIMULA 67 Common Base Language*. Norwegian Computing Center, Oslo.

[6] Douglass, Bruce Powell (1998). Real-Time UML - Developing Efficient Objects for Embedded Systems. Addison-Wesley.

[7] Fowler, Martin and Scott, Kendall (1997). UML Distilled - Applying the Standard Object Modeling Language. Addison-Wesley.

[8] Gosling, James and Joy, Bill and Steele, Guy (1996). *The Java Language Specification*. Addison-Wesley.

[9] Kaelbling, Leslie Pack (1993). Learning in Embedded Systems. MIT Press.