Anul LI 2006

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

Redacţia: 3400 Cluj-Napoca, str. M. Kogălniceanu nr. 1 Telefon 405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

NONLINEAR EVOLUTIONARY SUPPORT VECTOR MACHINES. APPLICATION TO CLASSIFICATION

RUXANDRA STOEAN, D. DUMITRESCU, AND CATALIN STOEAN

Abstract. Support vector machines are a modern and very efficient learning heuristic. However, their internal engine relies on not very easy or common mathematical concepts. The paper presents a newly developed simpler design of the engine, built through the means of evolutionary computation, in the context of nonlinear support vector machines. Experiments are carried on fictitious 2-dimensional points data sets and demonstrate once again the promise of the new approach.

Keywords:*support vector machines, nonlinear hyperplane, evolutionary algorithms, polynomial classifier, radial decision function, 2-dimensional points data sets*

1. Introduction

Support vector machines (SVMs) are a type of learning machines [9]. According to [6], "support vector machines are a system for efficiently training linear learning machines in kernel-induced feature spaces, while respecting the insights of generalization theory and exploiting optimization theory".

As all learning machines, SVMs act in two stages. In the training stage, the correspondence between every input vector and given output is internally discovered and learnt. In the test step, prediction of the output for previously unknown input vectors is performed according to what has been learnt.

SVMs have been successfully applied to a wide range of pattern recognition problems. Interest in present paper is shown however only in what concerns classification.

Received by the editors: December 1, 2005.

2000 *Mathematics Subject Classification.* 68T05, 68T20, 92D10.

1998 *CR Categories and Descriptors.* I.2.6 [**Learning**]: Subtopic – *Concept Learning*; I.2.8 [**Problem Solving, Control Methods, and Search**]: Subtopic – *Heuristic methods*; J.3 [**Life and Medical Sciences**]: Subtopic – *Biology and Genetics*.

The task for SVMs here is then to achieve an optimal separation of data into classes. By resorting to evolutionary algorithms, authors propose a simpler alternative to the standard approach of SVMs to solving this optimization problem.

Initially, SVMs were developed for linearly separable data and were later improved to handle nonseparable cases as well. Consequently, the new technique, called evolutionary support vector machines (ESVMs), has followed the same steps. Linear separating hyperplanes for separable and nonseparable data were detected through evolutionary computation in [11, 12]. The last and most difficult step, i.e. the discovery of the optimal nonlinear hyperplane to deal with nonseparable data, is treated in present paper. The data sets that experiments are conducted on are fictitious 2-dimensional points sets.

The paper is structured as follows. Section 2 presents an overview of support vector machines for classification. Section 3 outlines the idea and structure of evolutionary support vector machines; values for parameters of both the evolutionary algorithm and the support vector machine are appointed and conducted experimental results are illustrated. Finally, some conclusions are reached and ideas for future work are discussed.

2. Principles of support vector machines

In standard manner, support vector machines deal with binary classification problems. They have, however, been extended to handle multi-class categorization. The new evolutionary support vector machines are built according to the classical binary situation; in the future, ESVMs will also be broadened to cover the multi-class circumstances. Consequently, in what follows, the concepts within SVMs will be explained on binary labelled data [3, 10].

Suppose the training data is of the following form:

$$\{(x_i, y_i)\}_{i=1,2,\dots,m} \tag{1}$$

where every $x_i \in R^n$ represents an input vector and each y_i an output (label).

Let us first suppose that the two subsets of input vectors labelled with $+1$ and -1, respectively, are linearly separable. The positive and negative training vectors are then separated by the hyperplane:

$$\langle w, x \rangle - b = 0, \tag{2}$$

where $w \in R^n$ is the normal to the hyperplane, $b \in R$ and $\frac{|b|}{\|w\|}$ is the distance from the origin to the hyperplane.

Accordingly, two data subsets are linearly separable iff there exist $w \in R^n$ and $b \in R$ such that:

$$\begin{cases} \langle w, x_i \rangle - b > 0, & y_i = 1, \\ \langle w, x_i \rangle - b < 0, & y_i = -1, i = 1, 2, ..., m. \end{cases} \tag{3}$$

According to [1], two data subsets are linearly separable iff there exist $w \in R^n$ and $b \in R$ such that:

$$\begin{cases} \langle w, x_i \rangle - b > 1, & y_i = 1, \\ \langle w, x_i \rangle - b < -1, & y_i = -1, i = 1, 2, ..., m. \end{cases} \tag{4}$$

Consequently, the separating hyperplane lies in the middle of the parallel supporting hyperplanes of the two classes.

Following the structural risk minimization principle [13, 14, 15], i.e. one gets that, in order to generalize well, the support vector machine must provide a hyperplane that separates the training data with as few errors as possible and, at the same time, with a maximal margin of separation. One subsequently obtains the optimization problem (P_1):

$$\begin{cases} \text{find } w \text{ and } b \text{ as to minimize } \frac{\|w\|^2}{2}, \\ \text{subject to } y_i(\langle w, x_i \rangle - b) \geq 1, i = 1, 2, ..., m. \end{cases} \tag{5}$$

where $\frac{2}{\|w\|}$ is the value of the margin.

Given a training data set that is nonseparable, it is obviously not possible to build a separating hyperplane without any classification errors. However, construction of an optimal hyperplane that minimizes misclassification would be of interest [8]. Previous ideas can be extended to handle this new situation by relaxing the constraints in (4). This can be achieved by bringing in some positive variables, called *slack* variables [4]. The introducing of these new variables relies on the fact that any training data point has a deviation from its supporting hyperplane, i.e. from the ideal condition of data separability, of $\frac{\pm\xi_i}{\|w\|}$. This affects the separation condition, which then becomes [4]:

$$y_i(\langle w, x_i \rangle - b) \geq 1 - \xi_i, i = 1, 2, ..., m \tag{6}$$

where $\xi_i \geq 0$. Thus, for a training data point to be erroneously classified, its corresponding ξ_i must exceed unity.

Simultaneously with (6), sum of misclassifications must be minimized. As a consequence, (P1) changes to (P2):

$$\begin{cases} \text{find } w \text{ and } b \text{ as to minimize } \frac{\|w\|^2}{2} + C\sum_{i=1}^{m} \xi_i, \\ \text{subject to } y_i(\langle w, x_i\rangle - b) \geq 1 - \xi_i, \\ \xi_i \geq 0, i = 1, 2, ..., m. \end{cases} \tag{7}$$

where C corresponds to assigning higher penalties for errors.

The concepts can be extended even further to the construction of a nonlinear separating hyperplane for nonseparable data. Based on [5], the training data can be nonlinearly mapped into a high enough dimensional space and linearly separated there.

Suppose an input vector is mapped into some Euclidean space, H, through a mapping $\Phi : R^n \mapsto H$. It can be easily seen that within (P_2) vectors in R^n appear only as part of dot products. Vectors in H should appear as part of dot products in its formulation, as well. Therefore, the equation of the separating hyperplane in H becomes:

$$\langle \Phi(w), \Phi(x_i)\rangle - b = 0 \tag{8}$$

where $\Phi(w)$ is the normal to the hyperplane.

The squared norm

$$\|w\|^2 = \langle w, w\rangle \tag{9}$$

changes to

$$\langle \Phi(w), \Phi(w)\rangle. \tag{10}$$

Now, if there were a kernel function K such that:

$$K(x, y) = \langle \Phi(x), \Phi(y)\rangle \tag{11}$$

where $x, y \in R^n$, one would use K in the training algorithm and would never need to explicitly even know what Φ is.

At this moment, the question is what kernel functions meet (11). The answer is given by Mercer's theorem from functional analysis [2]. The problem is that it may not be easy to check whether Mercer's condition is satisfied in every case of a new kernel. There are, however, a couple of classical kernels that had been demonstrated to meet Mercer's condition [2]:

- Polynomial classifier of degree p: $K(x, y) = \langle x, y\rangle^p$
- Radial basis function classifier: $K(x, y) = e^{\frac{\|x-y\|^2}{\sigma}}$

Consequently, the optimization problem (P_2) will now change to (P3):

$$\begin{cases} \text{find } w \text{ and } b \text{ as to minimize } \frac{K(w,w)}{2} + C\sum_{i=1}^{m}\xi_i, \\ \text{subject to } y_i(K(w,x_i) - b) \geq 1 - \xi_i, \\ \xi_i \geq 0, i = 1, 2, ..., m. \end{cases} \tag{12}$$

3. Design of nonlinear evolutionary support vector machines

The optimization problem in support vector machines is standardly solved using concepts of convexity and resorting to an extension of the well-known method of Lagrange multipliers. The mathematics of the method can be found to be difficult.

Authors have brought the ESVM alternative to this technique, which is very easy to understand and apply. Standard evolutionary algorithms are used in this respect.

In present work, nonlinear ESVMs are outlined only. Linear ESVMs for separable and nonseparable data are particular situations of proposed algorithm; however, they can be found in [11, 12], respectively.

The way in which components of the evolutionary algorithm are considered with respect to nonlinear support vector machines is outlined. Experimental results are reached and illustration of the separation is given for different fictitious 2D points data sets appointed in order to obtain three nonlinear separating hyperplanes, *i.e* odd or even polynomial and radial classifiers.

3.1. **Components of the evolutionary algorithm.**

Components regard representation, initialization of the population, expression of the fitness function, the selection and variation operators.

Representation

A chromosome has the following structure of w, b and ξ:

$$c = (w_1, ..., w_n, b, \xi_1, ..., \xi_m) \tag{13}$$

Proposed evolutionary algorithm thus includes the training errors in the structure of the chromosome. In the end of the algorithm, the training points that are correctly placed will have the corresponding $\xi_i s$ less than unity, while those erroneously placed will have their $\xi_i s$ exceed it.

Initial population

Chromosomes are randomly generated following a uniform distribution, such that $w_i \in [-1, 1], i = 1, 2, ..., n$, $b \in [-1, 1]$ and $\xi_j \in [0, 1], j = 1, 2, ..., m$.

Fitness evaluation

The expression of the fitness function is considered as follows:

$$f(c) = f(w_1, ..., w_n, b, \xi_1, ..., \xi_m) = K(w, w) + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}[t(y_i(K(w, x_i) - b) - 1 + \xi_i)]^2, \quad (14)$$

where

$$t(a) = \begin{cases} a, & a < 0, \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

One is led to minimize(f(c), c).

Genetic operators

Tournament selection is used. Intermediate crossover and mutation with normal perturbation are considered. Mutation is restricted only for errors, preventing the ξ_is from taking negative values.

Stop condition

The algorithm stops after a predefined number of generations. In the end, it obtains the equation of the hyperplane, i.e. w and b. Errors on training set also result from the algorithm, i.e. those corresponding to $\xi_i > 1$, $i = 1, 2, ..., m$

3.2. **Experimental results.** Three fictitious 2-dimensional points data sets were built in order to allow construction of an even polynomial classifier, an odd one and a radial decision function. Illustration of their configurations and of the obtained nonlinear hyperplanes is given in Figures 1, 2 and 3.

Values for the specific parameters of every chosen kernel are given in turn in Table 1. The parameters of the support vector machine and of the evolutionary algorithm that are common to all kernels had the same values in all three cases and are outlined in Table 2. Some abbreviations are used therein, i.e. *ps* stands for population size, *ng* for number of generations, *cp* for crossover probability, *mp* for mutation probability, *ms* for mutation strength.

Degree odd polynomial	15
Degree even polynomial	14
Sigma	500

Table 1. Values for parameters of the chosen kernels

C	ps	ng	cp	mp	mp - $\xi_i s$	ms	ms - $\xi_i s$
1	200	1000	0.3	0.5	0.5	0.1	0.1

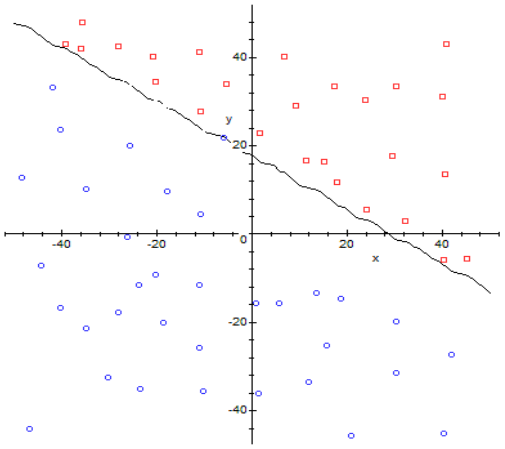


FIGURE 1. An odd polynomial classifier

Table 2. Values for parameters of the support vector machine and of the evolutionary algorithm

4. Conclusions and future work

From the discussion above, there arise various advantages in using evolutionary support vector machines instead of the classical architecture:

1. The evolutionary approach is much easier for both the developer and the end user.

2. The evolutionary solving of the optimization problem leads to the obtaining of w and b directly, while in the classical approach the equation of the optimal hyperplane is determined after Lagrange multipliers are found.

3. Moreover, in the case of the classical technique, when using kernels in which Φ cannot be explicitly obtained, it is not possible to determine w and b at all - it can only predict the class for a test vector.

4. The evolutionary method can also provide which training data cannot be correctly classified, as errors are included in the structure of the chromosomes; the evolutionary support vector machines self-determine their training error.

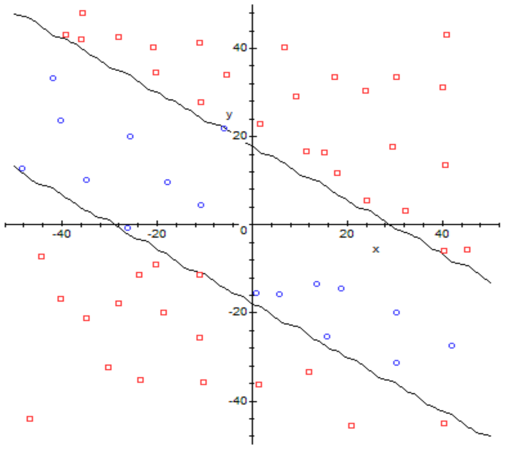


Figure 2. An even polynomial classifier

Future work envisages application and validation of proposed ESVMs to real-world problems. Also, the design of evolutionary multi-class support vector machines, based on classical SVM approaches to classification with more than two categories, is desired.

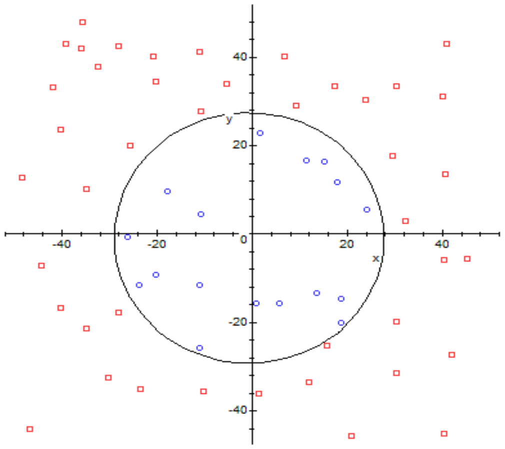


FIGURE 3. A radial polynomial classifier

REFERENCES

[1] R.A. Bosch, J.A. Smith, *Separating Hyperplanes and the Authorship of the Disputed Federalist Papers*, American Mathematical Monthly, Volume 105, Number 7, pp. 601-608, 1998

[2] B. E. Boser, I. M. Guyon and V. Vapnik, *A Training Algorithm for Optimal Margin Classifiers*, In D. Haussler, editor, Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, pp. 11-152, Pittsburgh, PA, ACM Press, 1992

[3] C.J.C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery 2, 121-167, 1998

[4] C. Cortes, V. Vapnik, *Support Vector Networks*, Machine Learning, 20:273-297, 1995

[5] T. M. Cover, *Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition*, IEEE Transactions on Electronic Computers, vol. EC-14, pp. 326-334, 1965

[6] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 2000

[7] D. Dumitrescu, B. Lazzerini, L.C. Jain, A. Dumitrescu, *Evolutionary Computation*, CRC Press, Boca Raton, Florida, 2000

[8] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, New Jersey, 1999

[9] R. Lothian, *Introduction to Support Vector Machines*, Talk, 2003

[10] B. Scholkopf, *Support Vector Learning*, Dissertation, Berlin, 1997

[11] R. Stoean, D. Dumitrescu, *Linear Evolutionary Support Vector Machines for Separable Training Data*, Annals of the University of Craiova, Seria Matematica-Informatica, submitted for publication, 2005

[12] R. Stoean, D. Dumitrescu, *Evolutionary Support Vector Machines - a New Learning Paradigm. The Linear Non-separable Case*, Proceedings of the Symposium "Colocviul Academic Clujean de Informatica", accepted for publication, 2005

[13] V. Vapnik, *Inductive Principles of Statistics and Learning Theory*, In Smolensky, Mozer and Rumelhart (Eds.), Mathematical Perspectives on Neural Networks, Lawrence Erlbaum, Mahwah, NJ, 1995

[14] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, New York, 1995

[15] V. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998

Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova, Craiova Romania
E-mail address: `ruxandra.stoean@inf.ucv.ro`

Faculty of Mathematics and Computer Science, Department of Computer Science, Babes-Bolyai University, Cluj - Napoca Romania
E-mail address: `ddumitr@cs.ubbcluj.ro`

Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova, Craiova Romania
E-mail address: `catalin.stoean@inf.ucv.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

AUTOMATA-BASED COMPONENT COMPOSITION ANALYSIS

ANDREEA FANEA, SIMONA MOTOGNA AND LAURA DIOŞAN

ABSTRACT. Formal specification of software components enables automatic composition and checking of component-based systems. The component system is modeled as a finite automaton. We propose an algorithm that builds all syntactically correct finite automata-based models of a component-based system. The result systems are checked against the properties: lost data and number of provider/inport.

1. Introduction

Component-based software engineering (CBSE) is the emerging discipline of the development of software components and the development of systems incorporating such components. In order to construct a correct system, these components should be assembled in an unified model and we would like to be able to obtain properties of the model that could contribute to its correctness.

A formal model for component-based software is of critical importance because it provides a basis for the understanding of the underlying concepts of component models, component certification techniques, component testing. The general definition of a software component is given in [5].

There are two issues which need to be addressed [1], [6] where a software system is to be constructed from a collection of components:

- Component integration - the mechanical process of wiring components together. There has to be a way to connect the components together.
- (Behaviour) Component composition - we have to get the components to do what we want. We need to ensure that the assembled system does what is required. Component integration is taken one step further to ensure that assemblies can be used as components in larger assemblies.

To achieve integration, syntactical composition is studied. It offers the necessary tool to meet for the requirements for wiring components together. Component integration is a more complex process which will need to assign also semantic

Received by the editors: 9.01.2006.

2000 *Mathematics Subject Classification.* 68N30, 68Q45.

1998 *CR Categories and Descriptors.* code I.6.4 [**Simulation and modeling**]: – *Model Validation and Analysis*; code I.6.5 [**Model Development**]: – *Modeling methodologies*.

information regarding behaviour to the syntactic entities. This will be the next step in our study and it is not treated here.

We have developed in [2] an algorithm that computes all the possibilities of constructing a system from a given set of components, checking only the syntactical part when wiring together the components. This computation is the first step from the construction of a component-based system. The next step consist of the behaviour composition of components.

In [4] a model of a component-based software system is proposed, which uses a finite automata-based method, enabling compositional reachability analysis. The following checks were performed:

- the system is consistent: starting from a given input, all components can be added to the model and the execution eventually terminates;
- there are no potential deadlocks in the model.

This paper proposes a new algorithm to construct all the component-based software systems as finite automata-based models. The resulting models are syntactically correct. By syntactically correct model we mean no semantic involvement in the models, but just the way to connect the components together, the mechanical process of "wiring" components together (component integration).

A syntactically correct model has the following properties:

- all inputs are provided for a component to be executed; a component is added into the model if and only if all its necessary input data are provided;
- there are no cyclic dependencies;
- no duplicate components.

The algorithm checks model consistency during its construction from a given set of components. The result systems are checked against the properties: if the execution of the component system is terminated and even if the system behaves properly, there is some lost data, and a component is not allowed to receive the value for an inport from more than one component - one provider/inport. A comparison analysis of three solutions (with different properties) are presented and some examples are given.

2. Previous results

In [4] the component system is modeled as a finite automaton, where components are represented as states and information flows as transitions.

Definition 1. *A source component, i.e. a component without inports, is a component that generates data provided as outports in order to be processed by other components.*

Definition 2. *A destination component, i.e. a component without outports, is a component that receives data from the system as its inports and usually displays it, but it does not produce any output.*

Definition 3. *A system of components is defined as a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, where:*

- *Q is the set of states, each $q \in Q$ representing a component;*
- *Σ is the input alphabet; in the proposed model, Σ is the union of the outports (of the components) already included in the models, in fact, the data set;*
- *$\delta : Q \times \Sigma \to P(Q)$ is the transition function; δ members have the form $((C_1, d) \to C_2)$, where C_1,$C_2 \in Q$ and $d \in outports(C_1) \bigcap inports(C_2)$;*
- *$q_0 \in Q$ is the initial state - the source component in the component system;*
- *$F \subset Q$ is the set of final states - the destination components from the component system.*

In [4] the *MakeModel* algorithm has as input a component system specification and builds the model, a nondeterministic finite automaton. The algorithm generates such a model from a given component system specification, checking the following properties:

- all inputs are provided for the tasks of the C component to be executed, i.e. $inports(C) \in \Sigma$;
- there are no "cyclic" component dependencies: C_1 expects data d_1 as inport and provides d_2 as outport, and component C_2 needs d_2 as inport and provides d_1 as outport.

In [4] the procedure *Search(compList, cond, component, flag)* searches into the list of components *compList* for the first component satisfying a given condition *cond*. The output parameter *flag* is set to true if the search is successful. In this case, the component is also provided. If no component matching *cond* is found then *flag* is set to false. Because of the "first" criterion, only one nondeterministic finite automaton is constructed.

In [3] the following definition was introduced:

Definition 4. *a. A component C is reachable iff there exists a path from the source component to C. We say that C' is reachable from C through d if $\delta(C, d) = C'$;*
b. A data d is live iff for a reachable component C there exists a component C‘ reachable from C through d: $d \in outports(C) \cap inports(C')$.

We have modified the algorithm [4] in order to generate all the nondeterministic finite automata. Also, the final constructed system have only live data. This property is checked after building the consistent system (starting from a given input, all components are added to the model and the execution eventually terminates). The construction of the model is described in the following section.

3. Model building

We must first establish our entities involved in the component system definition:

- domain D - a set that does not contain the null element;
- set of attributes A - an infinite fixed and arbitrary set; the atributes signify variables or fields;
- type of an attribute $x \in A : Type(x) \in D$ represents the set of possible values for the attribute x.

Consider the component system $CS = \{C_1, C_2, ..., C_n\}$, in which every component C_k is specified as: $C_k = (compID_k, inports_k, outports_k, functs_k)$, where:

- $compID_k$ is the component identifier, unique;
- $inports_k \subseteq A$ the set of input ports;
- $outports_k \subseteq A$ the set of output ports;
- $functs_k$ the set of tasks the C_k component performs.

3.1. Algorithm specification.

The specification of the **MakeAllModels** algorithm is as follows:

Begin

Input: the component system CS;

Output: all the nondeterministic finite automata A = (Q, Σ, δ, q_0, F).

End.

3.2. Algorithm description.

We use a recursive backtracking algorithm to generate all the component-systems from the existing specified components.

The first component that is used from the component system is a source component. A component is added to solution (the intern conditions, specified in ***valid(i)***) if the component was not already used before and all the inputs of the component are provided for the tasks to be executed. A component-based system is found (the conditions for the complete solution, specified in ***solution(i)***) when the last component added to solution is a destination component.

The lost data property is checked only after a solution is generated, because when integrating a component into the systems we do not check if all the outputs are consumed (only some outputs are used for the transition to the current component). It is obvious that for a component in the solution all the inputs are consumed, because we used the condition that all the inputs are available. The property checks if all the outputs of all the components involved into the computation are consumed.

The necessity to provide all inputs of the component to be executed generates another condition to be checked after a solution is generated: the inputs of a component could be provided by more than one component and the choice is made in the algorithm. The following situation is not desired: the current component

Algorithm 1 BuildingAllModels Algorithm

```
1: for each component in the system do
2:   add the component to the solution on position $i$;
3:   mark the new set of available inputs;
4:   if valid(i) then
5:     for each $mC \in CS$ do
6:       for each $d \in inports(Component_i)$ do
7:         if $d \in outports(mC)$ then
8:           $\delta := \delta \bigcup \{(mC, d) \rightarrow Component_i\}$;
9:         end if
10:      end for
11:    end for
12:    if not solution(i) then
13:      BuildingAllModels(i+1,...)
14:    else
15:      WriteSolution(i,...);
16:    end if
17:  end if
18: end for
```

receives the value for one of its inports from two different components. The algorithm will check at the end if a solution contains such situations. More precise explanations are presented in the next section.

4. Examples and result analysis

4.1. **Example 1.** Consider the following general set of components:

$C_1 = (C1, \varnothing, \{d_1, d_2\}, \{read\})$;
$C_2 = (C2, \{d_1, d_3\}, \{d_5, d_6\}, \{task_1, task_2, task_3\})$;
$C_3 = (C3, \{d_2\}, \{d_3, d_7\}, \{task_4\})$;
$C_4 = (C4, \{d_5, d_7\}, \{d_8\}, \{task_5\})$;
$C_5 = (C5, \{d_6, d_8\}, \varnothing, \{write\})$;
$C_6 = (C6, \{d_1, d_3\}, \{d_4, d_5, d_6\}, \{task_1, task_2, task_3\})$;
$C_7 = (C7, \{d_1, d_5\}, \{d_3\}, \{task_1, task_2, task_3\})$;
$C_8 = (C8, \{d_2, d_3\}, \{d_4\}, \{task_4\})$;
$C_9 = (C9, \{d_4\}, \{d_5, d_6\}, \{task_5\})$;
$C_{10} = (C10, \{d_6\}, \varnothing, \{write\})$;

The results of building the models from existing components are presented in Table 1: the number of all the consistent solutions, the number of solutions without lost data and the number of solutions with only one provider/input for all involved components and the number of final solutions. The percentage shows that only a small part of the solutions should be taken into consideration based on the efficiency criterion.

TABLE 1. The number of solutions for the components set from example 1

Algorithm MakeAllModels	All Solutions	Solutions without Lost Data	Solutions one provider/input	Final Solutions
Number	1323	40	64	1
Percent	100%	3.02%	4.83%	0.07%

The presented solution from Figure 1.a has lost data. The $C6$ component has two outports that are not "consumed".

The solution is $A = (Q, \Sigma, \delta, q_0, F)$, where:

- Q={C1,C3,C2,C4,C6,C5}
- Σ ={d1,d2,d3,d7,d5,d6,d8,d4}
- δ={(C1,d2) $\rightarrow$ C3,(C1,d1)$\rightarrow$ C2,(C3,d3)$\rightarrow$ C2, (C2,d5) $\rightarrow$ C4,(C3,d7)$\rightarrow$ C4,(C1,d1)$\rightarrow$ C6,(C3,d3)$\rightarrow$ C6, (C2,d6)$\rightarrow$ C5, (C4,d8)$\rightarrow$ C5,(C6,d6)$\rightarrow$ C5}
- q_0={C1}
- F={C5}

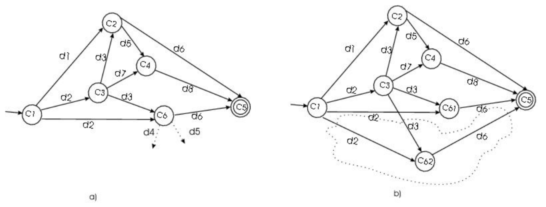


FIGURE 1. The nondeterministic finite automaton for the consistent solution a) with lost data d_4 and d_5 b) the correct model.

As Figure 1.a shows, component $C6$ outport contains data $d4$ and $d5$ which are lost (no other component from the system is using it). So we will split this component into two new components $C61$ and $C62$, clone its inports, data $d2$ and data $d3$, and then isolate the area containing component $C62$ and data $d2$, $d3$. The resulting model, as presented in Figure 1.b, is correct.

The presented solution is not included in the solution set with one provider/inport because the are two transitions to the same component $C5$ with the label $d6$ as in Figure 2. The input $d6$ of the $C5$ component must have only one provider on an execution. The "reverse" propagation of data (the output data of a component is propagated to two or more components) is allowed. Component $C3$ distributes the data $d3$ to $C2$ and $C6$ component as in Figure 2.a. In Figure 2.b the final solution is presented: the solution is consistent, no lost data and each inport for each component has just one provider.

4.2. **Example 2.** Consider the following general set of components:

$C_1 = (C1, \varnothing, \{d_1, d_2, d_3\}, \{read\});$
$C_2 = (C2, \{d_3\}, \{d_1\}, \{task_1, task_2, task_3\});$
$C_3 = (C3, \{d_3, d_4\}, \{d_5\}, \{task_4\});$
$C_4 = (C4, \{d_2\}, \{d_4\}, \{task_5\});$
$C_5 = (C5, \{d_5\}, \varnothing, \{write\});$
$C_6 = (C6, \{d_1\}, \{d_3\}, \{task_1, task_2, task_3\});$

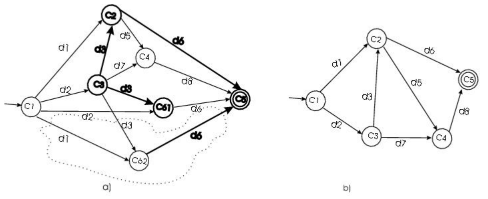


FIGURE 2. The finite automaton a) with more than one provider/inport; b) the corresponding final consistent solution.

The results of building the models from existing components are presented in Table 2.

TABLE 2. The number of solutions for the component set of Example 2

Algorithm MakeAllModels	All Solutions	Solutions without Lost Data	Solutions one provider/input	Final Solutions
Number	19	5	5	0
Percent	100%	26.31%	26.31%	0%

Figure 3 presents two solutions, one from the solution set without lost data figure 3a) and the other from the one provider/input solution set figure 3b). The consistent system $CS = \{C1, C4, C6, C3, C5\}$ from the 1a) side contains an output that is not "consumed": the data $d3$ is lost. On the right hand side there is a consistent system that has a component with more than one provider for an input: component $C3$ receives the data $d3$ from component $C1$ and from $C6$.

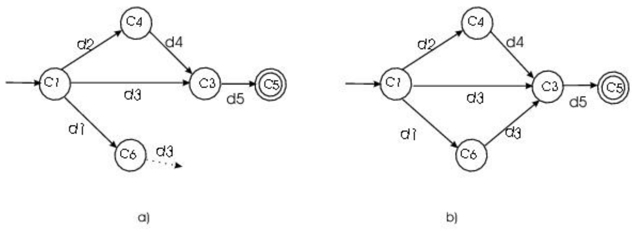


FIGURE 3. A finite automaton for the consistent solution a) without lost data; b) with more than one provider/inport.

5. Conclusions and future work

In this paper we proposed a new algorithm for computing all the component-based systems as automata-based models from a set of specified components. We analyse the final models from distinctive perspective, checking the existence of the properties lost data and one provider/port.

Using the same component model we intend to extend the algorithm and to address the following topics in the future: checking if the model supports a given sequence of tasks and building a component-based system that contains a given sequence of tasks. Also checking the behaviour of components (to ensure that the assembled system does what is required) after syntactic composition is intended to be studied.

References

[1] Ivica Crnkovic and Magnus Larsson, *Building Reliable Component-Based Software Systems*, Artech House publisher, 2002

[2] A. Fanea, S. Motogna, *A Formal Model for Component Composition*, Proceedings of the Symposium "Zilele Academice Clujene", 2004, pp. 160-167

[3] S. Motogna, B. Parv, D. Petrascu, Finding Errors in a Component Model Using Automata Techniques, 5th Joint Conference on Mathematics and Computer Science, Debrecen, Hungary, 2004, pp. 69

[4] B. Parv, S. Motogna, D. Petrascu, Component System Checking Using Compositional Analysis, Proceedings of the International Conference on Computers and Communications, 2004, Baile Felix Spa-Oradea, Romania, 2004, pp. 325-329

[5] Szyperski C. et al., Component Software, Beyond Object-Oriented Programming, 2nd ed., ACM Press, Addison-Wesley, NJ, 2002.

[6] Robert John Walters, *A Graphically based language for constructing, executing and analysing models of software systems*, University of Southampton, Faculty of Engineering and Applied Science Electronics and Computer Science, PhD Thesis, 2002

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {afanea, motogna, lauras}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

A PROGRAMMING INTERFACE FOR MEDICAL DIAGNOSIS PREDICTION

GABRIELA ŞERBAN, ISTVAN-GERGELY CZIBULA, AND ALINA CÂMPAN

Abstract. The aim of this paper is to present a programming interface that can be used for assisting physicians in medical diagnosis. The interface provides an original diagnosis technique based on relational association rules and a supervised learning method. Using the designed interface, we made an experiment for cancer diagnosis; the precision of the diagnosis on our testing data was 90%. The main advantage of the proposed interface is that can be use in diagnosis for every disease, and, much more, can be simply extended, by adding new symptom types and new relations between symptoms for the given disease.

Keywords: Relational association rules, Programming, Interface, Supervised Learning.

1. Introduction

The purpose of this paper is to propose a technique for assisting medical diagnosis using relational association rules and to present a programming interface for medical diagnosis, using the proposed technique. The interface is meant to facilitate the development of software for identifying the probability of illness in a certain disease.

Much more, this interface can be simply extended by adding new symptom types for the given disease, and by defining new relations between these symptoms.

In our model, we have a set of patients identified by a set of symptoms of a certain disease. The symptoms types and values are unimportant in our approach, the user of the interface can simply define concrete symptoms for the current diagnosis task.

For this issue, the patients, their symptoms types, and the relations between their symptoms can be designed and implemented separately and then interconnected relatively easily in a standard, uniform fashion.

Received by the editors: March, 1, 2006.

2000 *Mathematics Subject Classification.* 68P15, 68T05, 68N19.

1998 *CR Categories and Descriptors.* H.2.8[**Computing Methodologies**]: Database Applications – *Data Mining*; I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*; D.1.5[**Software**]: Programming Techniques – *Object-Oriented Programming*;

2. Relational Association Rules

We extend the definition of ordinal association rules ([3]) towards *relational association rules.*

Definition 1. *Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of entities (records in the relational model), where each record is a set of m attributes, $(a_1, \ldots, a_m)$. We denote by $\Phi(r_j, a_i)$ the value of attribute a_i for the entity r_j. Each attribute a_i takes values from a domain D_i, which contains ε (empty value, null). Between two domains D_i and D_j can be defined partial relations, such as: less or equal ($\leq$), equal ($=$), greater or equal ($\geq$), etc. We denote by $\mathcal{M}$ the set of all relations defined. An expression $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}) \Rightarrow (a_{i_1}\ \mu_1\ a_{i_2}\ \mu_2\ a_{i_3} \ldots \mu_{\ell-1}\ a_{i_\ell})$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_m\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $\mu_i \in \mathcal{M}$ is a relation over $D_{i_j} \times D_{i_{j+1}}$, is an* ***relational association rule*** *if:*

a) *$a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together (are non-empty) in $s\%$ of the n records ; we call s the* ***support*** *of the rule,*
and

b) *we denote by $R' \subseteq R$ the set of records where $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together and $\Phi(r_j, a_{i_1})\ \mu_1\ \Phi(r_j, a_{i_2})\ \mu_2\ \Phi(r_j, a_{i_3})\ \ldots\ \mu_{\ell-1}\ \Phi(r_j, a_{i_\ell})$ is true for each record r_j din R'; then we call $c = |R'|/|R|$ the* ***confidence*** *of the rule.*

The users usually need to uncover interesting relational association rules that hold in a data set; they are interested in relational rules which hold between a minimum number of records, that is rules with support at least s_{min}, and confidence at least c_{min} (s_{min} and c_{min} are user-provided thresholds).

Definition 2. *We call a relational association rule in R* interesting *if its support s is greater than or equal to a user-specified minimum support, s_{min}, and its confidence c is greater than or equal to a user-specified minimum confidence, c_{min}.*

In [3] is given a discovery algorithm for binary ordinal association rules (rules between two attributes). We developed in [4] an algorithm, called *DOAR* (Discovery of Ordinal Association Rules), that efficiently finds all ordinal association rules, of any length, that hold over a data set. We have proved that the proposed algorithm is correct and complete. This algorithm can be used for finding relational association rules, as well. For implementing the main functionality of our interface we have used the *DOAR* algorithm.

3. Medical Diagnosis using Relational Association Rules

In this section we propose a supervised learning technique, based on finding relational association rules, called *MDRAR* (Medical Diagnosis using Relational

Association Rules). *MDRAR* determines the probability that a patient characterized by a set of symptoms suffers from a certain disease. The method works as follows.

Let us consider that our focus is a certain disease, denoted by D. For an appropriate diagnosis in the disease D, we consider a set of n relevant symptoms, S. Each symptom is an attribute a_i and has values from a domain D_i. In order to predict the probable diagnosis we perform a training step using two knowledge bases: a knowledge base containing the patients that suffer from the disease D (the ill patients), and a knowledge base containing the healthy patients. In both knowledge bases, each patient is characterized by a vector with components the values of all the symptoms from S. The knowledge bases of ill and healthy patients are the training data and are used in the training step of the algorithm (see Figure 1).

Let us assume that a n-dimensional vector describing the symptoms from S observed at a patient P is given as input to our algorithm. *MDRAR* determines the probability that the patient P suffers from the disease D, using the model learned in the training step. This is the prediction step of the algorithm (see Figure 1).

The main steps of our diagnosis technique are given in Figure 1:

Figure 1. The *MDRAR* technique.

(1) **TRAINING STEP**:
  (a) determine from the knowledge base with ill patients the set of association rules (R_1) having a minimum *support* and *confidence*;
  (b) determine from the knowledge base with healthy patients the set of association rules (R_2) having a minimum *support* and *confidence*;
(2) **PREDICTION STEP**: for each patient P for which we intend to predict the diagnosis, calculate the probability that P suffers from the disease D as the percentage of rules from R_1 verified by P and rules from R_2 not verified by P.

4. The programming interface

In this section we propose an API that allows a simple development of applications for medical diagnosis based on finding relational association rules. The API provides an uniform development for all these applications.

The main advantage of the interface is that the user can simply define, depending on the current disease, new types of symptoms and new types of relations between the symptoms, and the diagnosis prediction process remains unchanged. The interface is realized in JDK 1.5, and is meant to facilitate software development for assisting medical diagnosis.

There are seven basic entities (classes): `Patient` (defines a patient), `Symptom` (characterizes the patients), `SymptomType` (represents the type of a symptom), `Relation` (defines a partial relation between symptoms), `AssociationRule` (describes relational association rules between symptoms), `AssociationRuleGenerator` (responsible with generating relational association rules from the set of patients, based on their symptoms values and relations already defined) and `Diagnosis` (responsible with predicting the diagnosis for a given patient, using the *MDRAR* algorithm).

For designing the interface, we made an abstraction of the mechanisms for generating relational association rules, in order the interface to be useful for any kind of disease, symptoms and relations between symptoms. Much more, the patient entities are completely separated from the symptoms that characterize them (a `Patient` has to know nothing about its `Symptoms`, it has to know only about their behavior). Thus, we can easily change and add `Symptoms` characterizing the `Patients`, and `Relations` between `Symptoms`, without affecting the general diagnosis prediction process.

The `AssociationRuleGenerator` class is the main class of the interface and manages the process of finding relational association rules in the given set of `Patients` with respect to the given `Symptoms`. This class provides an operation that finds relational association rules in data, by implementing the *DOAR* algorithm (section 2).

The interface also provides:

- the class `Patients` that models a set of patients (the data set from which we want to extract relational association rules);
- the class `AssociationRules` that models a set of relational association rules;
- the class (`Relations`) that manages the set of relations between the symptoms (this class allows to manage dynamically the set of relations defined between symptoms).

For using the interface in a specific diagnosis prediction task, the user has only to:

- define specialized classes for the concrete symptom types (for example a class `SymptomInt` that extends the abstract class `SymptomType` if the symptoms are quantified by integer values);
- define specialized classes for the concrete relations between symptoms (for example a class `IntIntEqual` that extends the abstract class `Relation` if we want to describe the equality relation between two integer valued symptoms);
- construct the concrete set of patients.

All other mechanisms needed for generating the rules and predicting the diagnosis are provided by the classes from the interface.

In the following we present the skeleton of a diagnosis application. Let us assume that symptoms have integer values, and the only relation needed between Symptoms is "=".

- First, the user implements the class that defines the concrete symptom type.
  public class `SymptomInt` **extends** `Symptom`{...}.

- Second, the user implements the class that defines the concrete relation between the symptoms already defined.
  public class `IntIntEqual` **extends** `Relation`{...}.

In the same manner as above, the user can define as many symptom types and relations as are needed in the current diagnosis prediction task.

In the application class the user has to define a method that reads the data (patients) from an external device (file, database) and returns a set of patients (an instance of the `Patients` class) and to add (register) the concrete relations defined above to the set of relations `Relations`.

```
public class Application {
 public Application(){
  // The manager of relations adds a new concrete relation to its set of
  // relations
  RelationSet.addRelation(new IntIntEqual());
  // The application provides a method that constructs the set of ill patients
  Patients ill = readData();
  // An instance of an object AssociationRuleGenerator is created from the
  // Patient set created above
  AssociationRuleGenerator arg = new AssociationRuleGenerator(ill);
  // The association rule generator generates the set of association rules
  // having a minimum support and confidence
  double minimumSupport = 0.9;
  double minimumConfidence = 0.65;
  AssociationRuleSet illRules = arj.genAssociationRules(minimumConfidence);
  // The application provides a method that constructs the set of healthy
  // patients
  Patients healthy = readData();
  // An instance of an object AssociationRuleGenerator is created from the
  // Patient set created above
  arg = new AssociationRuleGenerator(healthy);
  // The association rule generator generates the set of association rules
  // having a minimum support and confidence
  AssociationRuleSet healthyRules=arj.genAssociationRules(minimumConfidence,
                                   minimumSupport);
  // An instance of a Diagnosis class is now created
  Diagnosis d = new Diagnosis(healthyRulles, illRules);
  // Now the object d can be used to predict the diagnosis a given patient
 }
}
```

Figure 2 shows a simplified UML diagram ([6]) of the interface, illustrating the hierarchy of classes. The figure illustrates the core of the interface and what is outside the core are the concrete classes that the user has to define, by extending the classes provided by the interface, in order to develop a diagnosis prediction application for a given disease. It is important to mention that all the core classes provided by the interface remain unchanged in all applications.

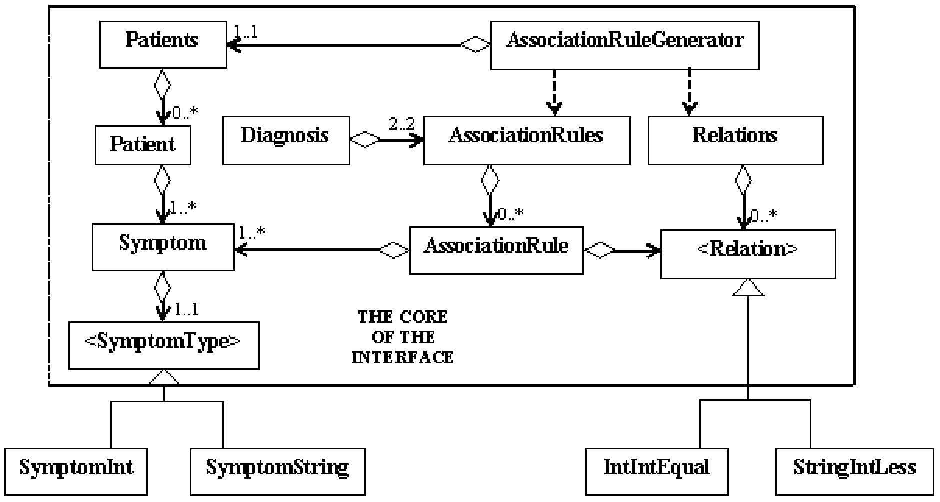


FIGURE 2. The diagram of the programming interface

5. The Design of the Interface

The classes used for realizing the interface are the following:

- `SymptomType` is `ABSTRACT`.

  Models an abstract symptom type, identified by the symptom type name. The class has operations for returning and modifying the symptom type name, a method for verifying the equality of two symptom types and an abstract method that creates a symptom value from a string. The concrete symptom types defined by the user of the interface, will extend the abstract SymptomType, managing their concrete type and overwriting the abstract methods from the abstract `SymptomType`;

- `Symptom`.
  Models a symptom characterizing the patients, and is characterized by a name, a symptom type (an instance of the `SymptomType` class) and a value (that is an object).
- `Patient`.
  Models a patient from the data set, which consists in a list of symptoms. The class has operations for managing the symptoms: adding, removing and returning symptoms from a given position, searching a symptom with a given name and symptom type.
- `Patients`.
  Models a set of patients, which consists in a list of *Patient* objects. The class has operations for managing the set of patients: adding, removing, searching patients and a method that returns an iterator on the set.
- `Relation` is `ABSTRACT`.
  Models an abstract relation between two symptom types $Type_1$ and $Type_2$. The class has abstract operations for: returning $Type_1$ and $Type_2$, returning the name of the relation, verifying if two `Symptoms` are in the given relation and for returning the converse of the relation.
- `AssociationRule`.
  Models a relational association rule, which consists in a set of abstract symptoms, a set of abstract relations, and characterized by its support and confidence. The main methods of this class are for: managing the symptoms and the relations from the association rule, setting and returning the support and the confidence of the rule.
- `AssociationRuleSet`.
  Models the structure of a set of relational association rules, which consists in a list of `AssociationRule` objects. The class has operations for managing the set of relational association rules: adding, removing, searching rules and a method that returns an iterator on the set.
- `Relations`.
  In our design this class models a repository (set) of relations, that allows the user to dynamically add relations between newly defined Symptom types. The user can dynamically add new defined relations in this list, using a method *addRelation*. This class has methods for obtaining the relations for a given `Symptom`, for verifying if there exists a given `Relation` between two `Symptoms`.
- `AssociationRuleGenerator`.
  Is the class that implements the process of finding relational association rules in a set of patients. The main method of the class is *generateAssociationRules*, that generates from the data set the relational

association rules having a minimum given support and confidence, and returns an instance of the `AssociationRuleSet` class.

- `Diagnosis`.

  Is the main class of the interface, that predicts a diagnosis for a given patient, based on the technique described in section 3. It represents the *heart* of the interface, the uniform usage that all patients, with their particular symptoms and relations, are meant to conform to. An instance of the `Diagnosis` class is associated with two instances of the `AssociationRuleSet` class.

As it can be seen on Figure 2, there is a dependency relationship between the `AssociationRuleGenerator` and `Relations`, that allows the association rule generator to dynamically manage the relations added by the user, without affecting the main process of detecting rules.

6. Experimental Evaluation

The file for this experiment was obtained from the website at "http://www.cormactech.com/neunet".

In order to test the above defined interface, we considered a HealthCare experiment for predicting the cancer disease.

The entities in this experiment are patients: each patient is identified by 9 Symptoms [1]. Each Symptom represents the value of a symptom in the cancer disease, and has integer values between 1 and 10. Each instance has one of 2 possible classes: benign or malignant. In this experiment are 457 patients (entities).

The attribute information used in the "cancer" experiment is shown in Table 1.

Table 1. Attribute information in the "cancer" experiment

	Attribute	Domain
1.	Clump Thickness	1 - 10
2.	Uniformity of Cell Size	1 - 10
3.	Uniformity of Cell Shape	1 - 10
4.	Marginal Adhesion	1 - 10
5.	Single Epithelial Cell Size	1 - 10
6.	Bare Nuclei	1 - 10
7.	Bland Chromatin	1 - 10
8.	Normal Nucleoli	1 - 10
9.	Mitoses	1 - 10

For this experiment, we have defined:

- `SymptomInt`, defining the integer Symptom representing a patient's symptom in the cancer disease;

- `IntIntEqual`, `IntIntLess` and `IntIntGreater`, defining the possible relations between two integer symptoms ($=$, $\leq$, $\geq$);
- a mechanism that reads the data (for each patient, it reads the values of the symptoms) and creates a `Patients` object.

We executed the medical diagnosis algorithm with minimum support threshold of 0.9 and minimum confidence threshold of 0.65. The training data consists in 147 ill patients and 260 healthy patients. The prediction step was made for 60 patients (30 ill patients and 30 healthy patients). We have obtained a precision of 90%.

As a conclusion of our experiments, we have to mention, from a programmer point of view, the advantages of using the interface proposed in this paper:

- is very simple to use;
- the effort for developing a diagnosis application based on relational association rule detection is reduced - we need to define only a few classes, the rest is provided by the interface;
- the user of the interface has to know nothing about the method of finding relational association rules or about the prediction method, because they are provided by the interface;
- we can add new symptom types and relations between symptoms, while the interface remains unchanged.

7. Conclusions and Further Work

As a conclusion, we have developed a small framework that will help programmers to build, dynamically, their own applications for diagnosis prediction in different kind of diseases, without dealing with the internal mechanism (that remains unchanged and is provided by the interface) and having the possibility to define their own types of symptoms and relations between symptoms. So, the programmer's effort for developing an application is small.

Further work can be done in the following directions:

- to test the accuracy of our technique on practical diagnosis. We think that increasing the number of data in the training step the accuracy of the prediction will grow;
- to study, for certain diseases, how can other symptoms and other relations between symptoms be added in order to assure better results in diagnosis.

References

[1] Wolberg, W., Mangasarian, O.L.: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193–9196.

[2] http://www.cormactech.com/neunet, "Discover the Patterns in Your Data", CorMac Technologies Inc, Canada.

[3] Marcus, A., Maletic, J. I., Lin, K.-I., "Ordinal Association Rules for Error Identification in Data Sets", CIKM 2001, 2001, pp. 589–591.

[4] Campan, A., Serban, G., Truta, T. M., Marcus, A., "An Algorithm for the Discovery of Arbitrary Length Ordinal Association Rules", submitted to DMIN'06.

[5] Han, J., Kamber, M., "Data Mining: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems, 2001.

[6] http://www.omg.org/technology/documents/formal/uml.htm.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: `gabis@cs.ubbcluj.ro`

"InfoWorld", Cluj-Napoca
E-mail address: `czibula.istvan@infoworld.ro`

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: `alina@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

RELATIONAL ASSOCIATION RULES AND ERROR DETECTION

ALINA CÂMPAN, GABRIELA ŞERBAN, AND ANDRIAN MARCUS

ABSTRACT. In this paper we introduce a new kind of association rules, relational association rules, which are an extension of ordinal association rules ([1]). The relational association rules can express various kinds of relationships between record attributes, not only partial ordering relations. We use the discovery of relational association rules for detecting errors in data sets. We report a case study for a real data set which validates this data cleaning approach and shows the utility of relational rules.

Keywords: Data Mining, Relational Association Rules, Data Cleaning.

1. Introduction

Association rule mining techniques are used to search attribute-value pairs that occur frequently together in a data set ([4], [5]).

Ordinal association rules ([1]) are a particular type of association rules. Given a set of records described by a set of attributes, the ordinal association rules specify ordinal relationships between record attributes that hold for a certain percentage of the records. However, in real world data sets, attributes with different domains and relationships between them, other than ordinal, exist. In such situations, ordinal association rules are not powerful enough to describe data regularities. Consequently, we define *relational association rules* in order to be able to capture various kinds of relationships between record attributes.

Discovering the ordinal rules that hold in a data set was already used for identifying possible errors in that data set ([1]). We apply relational association rules discovery to the same purpose. We provide an example that illustrates the utility

Received by the editors: March, 1, 2006.

2000 *Mathematics Subject Classification.* 68P15, 68U35.

1998 *CR Categories and Descriptors.* H.2.8[**Computing Methodologies**]: Database Applications – *Data Mining*; H.4.2[**Information Systems**]:Information Systems Applications – *Types of systems*;

of discovering relational rules in data. By using ordinal rules discovery not all types of errors that have been discovered using relational rules can be detected.

2. Relational Association Rules

We extend the definition of ordinal association rules ([1]) towards *relational association rules.*

Let $R = \{r_1, r_2, \dots, r_n\}$ be a set of entities (records in the relational model), where each record is a set of m attributes, $(a_1, \dots, a_m)$. We denote by $\Phi(r_j, a_i)$ the value of attribute a_i for the entity r_j. Each attribute a_i takes values from a domain D_i, which contains ε (empty value, null). Between two domains D_i and D_j can be defined relations, such as: less or equal ($\leq$), equal ($=$), greater or equal ($\geq$), etc. We denote by $\mathcal{M}$ the set of all relations defined.

Definition 1. A *relational association rule* is an expression $(a_{i_1}, a_{i_2}, a_{i_3}, \dots, a_{i_\ell}) \Rightarrow (a_{i_1}\ \mu_1\, a_{i_2}\ \mu_2\ a_{i_3} \dots \mu_{\ell-1}\ a_{i_\ell})$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \dots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \dots, a_m\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $\mu_i \in \mathcal{M}$ is a relation over $D_{i_j} \times D_{i_{j+1}}$, D_{i_j} is the domain of the attribute a_{i_j}. If:

a) $a_{i_1}, a_{i_2}, a_{i_3}, \dots, a_{i_\ell}$ occur together (are non-empty) in $s\%$ of the n records, then we call s the *support* of the rule,
and
b) we denote by $R' \subseteq R$ the set of records where $a_{i_1}, a_{i_2}, a_{i_3}, \dots, a_{i_\ell}$ occur together and $\Phi(r_j, a_{i_1})\ \mu_1\ \Phi(r_j, a_{i_2})\ \mu_2\ \Phi(r_j, a_{i_3})\ \dots\ \mu_{\ell-1}\ \Phi(r_j, a_{i_\ell})$ is true for each record r_j din R'; then we call $c = |R'|/|R|$ the *confidence* of the rule.

We call the length of a relational association rule the number of attributes in the rule. The length of a relational association rule can be at most equal to the number of the attributes describing the data.

The users usually need to uncover interesting relational association rules that hold in a data set; they are interested in relational rules which hold between a minimum number of records, that is rules with support at least s_{min}, and confidence at least c_{min} (s_{min} and c_{min} are user-provided thresholds).

Definition 2. We call a relational association rule in R *interesting* if its support s is greater than or equal to a user-specified minimum support, s_{min}, and its confidence c is greater than or equal to a user-specified minimum confidence, c_{min}.

We developed in [2] an algorithm, called *DOAR* (Discovery of Ordinal Association Rules), that efficiently finds all interesting ordinal association rules, of any length, that hold over a data set. This algorithm can be used for finding interesting relational association rules, as well.

3. Data Cleaning

Real-world data tend to be incomplete, noisy and inconsistent. Data cleaning refers to detect and correct or remove corrupt or inaccurate records (inconsistencies) from a record set, to fill in missing values or to smooth out noise while identifying outliers ([4]).

We aim to detect and report (not correct) record values that represent potential error in the analyzed data. We proceed in the same manner as for ordinal association rules discovery ([1]):

- We detect all the interesting binary relational rules (rules between two attributes), with respect to the user-provided support and confidence thresholds). Even if the *DOAR* algorithm can be used to discover all the relational rules, of any length, in a data set, we used it to discover only the binary rules. Binary rules are sufficient in order to detect errors in data sets.
- We detect and mark each record value that brokes any of the discovered binary relational rules.
- We report as potential errors those record values marked as possible errors more times than the average.

4. Case Study

For conducting our case study, we used a programming interface, presented in [3] and designed for the discovery of interesting relational association rules. This interface implements the *DOAR* algorithm. Based upon this interface, we developed an error detection application, following the steps described in section 3.

The data set we used in our case study consists in records containing information about students in a university department. There are 2012 records in the data set. Each record is described by the following attributes: StudentID - number, FirstName - text, LastName - text, CNP (Numerical Personal Code) - 13 digits number, BirthDate - date, RegistrationDate - date.

Between these attributes the following semantic relationships must hold: the CNP value must contain the BirthDate value and the BirthDate value must be earlier than the RegistrationDate value for every student record. We want to discover what are the erroneous records in the data set and which attribute value is most likely to be inconsistent with the rest of the record.

There are such data sets for which the semantic of some of the relationships between attributes describing the data are known. Exceptions from these known rules can be easily detected, but is more difficult, when other external information are not available, to establish which of the conflictual data are real errors. When other unknown regularities also exist in the data set, relational rule discovery, used as described in this paper, can help to estimate where an error resides.

We executed the *DOAR* algorithm with minimum support threshold of 0.95 and minimum confidence threshold of 0.93. The algorithm discovered that two binary interesting relational rules hold in the data set, as we expected:

```
CNP ‘‘=’’ BirthDate (support=0.970, confidence=0.937)
BirthDate ≤ RegistrationDate (support=0.970, confidence=0.967)
```

The difference between the support and confidence values of these two rules indicate that there are small irregularities in data, which represent potential errors. The average rules broken by the record values, as reported by our application, is 1.014. So, every record value that brokes both rules is reported as a potential error.

As the two binary rules discovered in data have only one common attribute, only this attribute values are reported as possible errors. Usually, when there are more relational rules having more common attributes, it is possible that errors to be detected at record values of different attributes.

We report below the potential errors found by our application.

```
s34 (1790521311822, May 24 1979, Jan 01 1978) :  2 errors at BirthDate
     Cnp(1790521311822) ‘‘=’’ BirthDate(May 24 1979);
     BirthDate(May 24 1979) ≤ RegistrationDate(Jan 01 1978);
s34 (1790521311822, May 24 1979, Jan 01 1978) :  2 errors at BirthDate
     Cnp(1790521311822) ‘‘=’’ BirthDate(May 24 1979);
     BirthDate(May 24 1979) ≤ RegistrationDate(Jan 01 1978);
s2572 (1771103062952, Nov 03 1997, Jan 01 1996) :  2 errors at BirthDate
     Cnp(1771103062952) ‘‘=’’ BirthDate(Nov 03 1997);
     BirthDate(Nov 03 1997) ≤ RegistrationDate(Jan 01 1996);
s2572 (1771103062952, Nov 03 1997, Jan 01 1996) :  2 errors at BirthDate
     Cnp(1771103062952) ‘‘=’’ BirthDate(Nov 03 1997);
     BirthDate(Nov 03 1997) ≤ RegistrationDate(Jan 01 1996);
```

5. Conclusions and Further Work

The concept of relational association rules, introduced in this paper, is a generalization of ordinal association rules. Relational rules discovery has a larger applicability, in different application domains where ordinal rules are not powerful enough to express all existing relationships between data attributes.

Further work can be done in the following directions:

- Defining relational association rules that contain repeating attributes; developing a technique similar to $DOAR$ for the discovery of such interesting rules.
- Applying discovery of relational association rules in other application domains, such as medical diagnosis.
- Using relational association rules of arbitrary length together with other data mining techniques such as classification or regression to increase the accuracy of the predictive models ([6]). Binary association rules are currently used in building predictive models in e-banking services ([7]).

References

[1] Marcus, A., Maletic, J. I., Lin, K.-I., "Ordinal Association Rules for Error Identification in Data Sets", CIKM 2001, 2001, pp. 589–591.

[2] Campan, A., Serban, G., Truta, T. M., Marcus, A., "An Algorithm for the Discovery of Arbitrary Length Ordinal Association Rules", submitted to DMIN'06.

[3] Serban, G., Campan, A., Czibula, I.G. , "A Programming Interface For Finding Relational Association Rules", submitted to ICCCC 2006.

[4] Han, J., Kamber, M., "Data Mining: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems, 2001.

[5] Tan, P.-N., Steinbach, M., Kumar, V., "Introduction to Data Mining", Addison Wesley, cap. 8,9, 2005.

[6] Hong, S. and Weiss, S., "Advances in predicitve model generation for data mining", IBM Research Report RC-21570.

[7] Aggellis, V., and Christodoulakis, D., "Association Rules and Predictive Models for e-Banking Services", in Proceedings of 1st Balkan Conference in Informatics, Tessaloniki, Greece, 2003.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: `alina@cs.ubbcluj.ro`

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: `gabis@cs.ubbcluj.ro`

Department of Computer Science, Wayne State University, USA
E-mail address: `amarcus@wayne.edu`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

SUPPORTING MULTIMEDIA STREAMING APPLICATIONS INSIDE THE NETWORK

ADRIAN STERCA, FLORIAN BOIAN, DARIUS BUFNEA, CLAUDIU COBÂRZAN

Abstract. There are many reasons for quality degradation of multimedia streams inside the network. Two of the most important ones are related to the UDP protocol and the Random-Drop policy implemented inside most of the routers currently deployed on the Internet. We intend to discuss, in this paper, some of multimedia streaming applications' problems centered around the UDP protocol and Random-Drop policy. We also present some mechanisms to alleviate these problems. More specifically, we present a queue management algorithm PDQMAMS (Priority-Drop Queue Management Algorithm for Multimedia Streams) for supporting the quality of multimedia streams inside the network.

1. Introduction

As multimedia encoding standards like MPEG and H.264/AVC become more efficient and as new digital video processing tools are developed, multimedia streaming applications gain a more important percent of the data transferred over the Internet. However, the current Internet does not support the high bandwidth and low latency demands of multimedia applications. Hence, communicating partners need to adapt to rapidly changing connection parameters and also to provide the best use of the scarce bandwidth of nowadays networks. Because of the latter reason, most of multimedia streaming applications choose to use UDP instead of TCP as the transport protocol, trading speed over reliability.

UDP offers several advantages over TCP, the most important being its speed. But the speed of UDP comes with a cost: it is unreliable and not responsive to congestions. Because UDP is unresponsive to congestion, it can cause the occurrence of congestion and can lead to unfairness against TCP-friendly flows. Most of the routers in the Internet use a Random Drop policy for freeing up space when congestion occurs and the queue is full. This policy gives multimedia packets an

Received by the editors: April 2, 2006.

2000 *Mathematics Subject Classification.* 90B18, 68M20.

1998 *CR Categories and Descriptors.* 90B18 [**Operations research, mathematical programming**]: Operations research and management science – *Communication networks*; 68M20 [**Computer science**]: Computer system organization – *Performance evaluation; queueing; scheduling*;

uniform importance although multimedia encoding schemes regard them as having different levels of priorities. This way of things is very bad for multimedia streaming applications. Because most of the encoding standards use layered/predictive encoding to achieve a good compression rate (i.e. some video frames are encoded as the difference between the current frame and the previous and/or the next one), dropping an independent frame in a router makes the other frames dependent on this one useless on the end-user's system even if they survive the ride through the Internet.

This paper presents a queue management algorithm for routers that tries to alleviate the aforementioned problems. It maintains a state for each multimedia flow and uses a priority-driven dropping scheme when the buffer overflows. Also, it is fair to each flow when packet drops are imperative and smart enough to drop packets belonging to less important frames, if congestion occurs. The rest of the paper is structured as follows: section 2 presents the main problems faced by multimedia streams inside the network and the causes of these problems; section 3 presents the AMSP protocol (Adaptation-aware Multimedia Streaming Protocol) whom our solution is based on and our own Priority-Drop Queue Management Algorithm for Multimedia Streams (PDQMAMS); in section 4 we perform experiments for proving PDQMAMS's qualities; section 5 presents some improvements of the algorithm we have in mind; then, in section 6 we refer to related work and the paper ends with conclusions and future work in section 7.

2. Problems of multimedia streams inside the network

Multimedia streaming application have huge requirements related to the network. One of the most important and hard to satisfy requirements is the high bandwidth that multimedia streaming applications need. For example, the MPEG-1 compression standard [1] demands a bandwidth of up to 1.5 Mbit/s, while the MPEG-2 standard [1] supports data rates of up to 4 Mbit/s. Besides huge bandwidth amounts, there are other requirements related to the timeliness of the transmission of multimedia streams. For an optimal quality of the multimedia data, real-time multimedia streaming often demands that communication be isochronous. This implies very short delays and also small values for jitter, but also good continuous throughput. If the delay between packets is too high, the video will periodically freeze at the receiver, waiting for the following frames to arrive. Jitter is equally important: a jitter too great means high fluctuations of the delay and this can lead to buffer underrun or overrun at the receiver, causing degradations of the video. These requirements are hard to satisfy in a best-effort network like the Internet.

The major problem of a best-effort network with respect to multimedia applications is that no QoS guarantees can be given. In a network with variable delays and different levels of congestion, communicating partners need to adapt to rapidly changing connection parameters (e.g. reducing the quality of the multimedia stream, *adapting* the video or audio material). Multimedia streaming applications often choose UDP as the transport layer protocol, instead of TCP, trading low delay and high throughput over reliable delivery. More specifically, multimedia streaming applications choose UDP over TCP to avoid TCP's start-up delay, to avoid the overhead of maintaining a state for each connection (like TCP does) and to favour timeliness characteristics (e.g., delay, jitter, etc.) at the cost of reliability (e.g., retransmission timeouts, in-order delivery, etc.). The speed advantage of UDP over TCP comes with a drawback: it is unresponsive to congestion. This is a major pitfall of multimedia streaming applications, as multimedia streams, which are basically UDP flows, are not fair to TCP-friendly flows (i.e. a flow whose sending rate does not surpass the sending rate of a TCP connection in the same circumstances [2]) and also they are not fair to each other. Depending on each connection's parameters, one UDP flow can eat up a lot more downstream bandwidth than the other flows that pass through the same router, especially if these are TCP-friendly flows. In steady-state, when a TCP flow notices congestion (a packet drop or ECN packet [3]) it backs-off, reducing its sending rate by half. In contrast, a UDP flow does not sense congestion because it is stateless and not connection-oriented like TCP. If packets are dropped, UDP flows continue to send packets at the same rate and, in the worse case, they can even increase their sending rate. This way, they can lead to starvation of other TCP-friendly flows [4] or even to congestion collapse [5]. A remedy for the congestion and unfairness problems of UDP flows can be considered from two perspectives. First, congestion avoidance and fairness with respect to other flows can be achieved if multimedia streams implement some form of end-to-end congestion control similar to the AIMD mechanism of TCP - this could be implemented either to the transport level (DCCP [6]) or to the application level -, so that the multimedia flow decreases its rate in response to congestion notification from the network. Second, the problems of congestion control and fairness between flows can be tackled inside the routers using active queue management like RED (Random Early Detection) [4, 8] or fair scheduling mechanisms like SFQ (Stochastic Fairness Queuing) [9] and WFQ (Weighted Fair Queuing) [10]. Active queue management algorithms try to avoid congestion and achieve fairness by managing the length of the queue and dropping packets from the queue when necessary or appropriate. On the other hand, scheduling algorithms decide which packet to send next and they mainly provide fair allocations rather than congestion avoidance.

The drop policy of most routers currently deployed on the Internet is the Random-Drop Policy. We refer with the term of "Random-Drop Policy" to all dropping policies that don't take into consideration the differences between packets when taking a dropping decision and the drop is made only when the queue overflows (e.g. Head-Drop, Tail-Drop, Random-Drop). The Random-Drop Policy is disastrous for multimedia streaming applications. The majority of multimedia encoding standards (e.g., MPEG, H.264/AVC, etc.) use a layered/predictive encoding scheme in order to achieve a good compression ratio, i.e., some of the frames depend on other *base frames* in the encoding process. In this encoding process, a basic layer is first encoded normally and any other enhancement layers are encoded as the difference between the base layer and the desired quality layer. For example, the MPEG-family of standards [11] uses four types of frames in the encoding process: *I-frames* (Intra-coded frames), *P-frames* (Predictive-coded frames), *B-frames* (Bi-directionally predictive-coded frames) and *D-frames* (DC-coded frames).

The Random-Drop Policy gives multimedia packets an uniform importance although multimedia encoding schemes regard them as having different levels of priorities. Dropping a packet belonging to a base layer (e.g. I-frame) makes subsequent packets/frames that depend on this one useless at the receiver, because it can not decode the enhancement layer (B- or P-frames) without the base layer (depending I-frame). A possible solution for this problem assumes assigning some sort of priorities (the priorities should be assigned either by the router or by the streaming protocol) to packets and drop them accordingly. In other words, we would rather drop an insignificant packet - one whose missing is tolerable by the receiver - than throw away indispensable data.

3. Priority-Drop Queue Management Algorithm for Multimedia Streams (PDQMAMS)

We argue that providing a real support for multimedia streams inside the network, at router level, requires some form of help from multimedia streaming protocols (priority schemes, feedback, etc.). The router algorithm, whether queue management algorithm or packet scheduling algorithm, can not do this job on its own. Following this direction, we present a router-queue management algorithm that uses knowledge from the multimedia streaming protocol (knowledge inserted in protocol's header inside packets) for preferentially dropping packets and, thus, for providing support for multimedia streams. We chose to use the AMSP protocol [12] instead of RTP [7], which is widely used for multimedia communication, because it conveys scaling information together with multimedia data as opposed to RTP which offers limited support for scaling. This is exactly what we need if we want to alleviate the impact of congestion on multimedia streams inside the network. More specifically, by assigning priorities to multimedia packets, AMSP conveys "intelligence" to network routers which would help them in taking better dropping decisions. AMSP also offers better QoS feedback support than RTP.

3.1. **The AMSP protocol.** The Adaptation-aware Multimedia Streaming Protocol is a streaming protocol similar to RTP. It conveys time sensitive information like multimedia data together with scaling information, so that multimedia streams can be adapted inside the network to the rapid changing parameters of the network. The scaling information can be used by common core routers to perform packet-level adaptation of multimedia streams (i.e. drop less important packets) or it can be used by scaling proxies that can perform complex media transformations inside the network (e.g. color reduction, temporal reduction, transcoding, etc.). The central concept of AMSP is the channel concept. Each channel is identified by an 8-bit field in the AMSP header called the ChannelID field. This field encapsulates the channel number, the priority of the channel and the dropping capability (whether packets belonging to this channel can be dropped). There a several types of channels AMSP supports: *control channel*, *media channels*, *metadata channels*, *scaling control channels*, *retransmission channels*, *feedback channels* and *auxiliary channels*. A multimedia stream is mapped onto one or more media channels, and thus, its packets get the priority of the respective channel(s). The idea of using channels to convey data with different levels of importance comes from the fact that multimedia streams are based on related layers with different levels of importance. AMSP offers elaborate feedback support including acknowledgements, non-acknowledgements, number of received/discarded/duplicated packets, round-trip time, jitter, bandwidth, buffer size, etc.

By assigning different levels of importance to multimedia packets, an AMSP-aware router could break this uniform treatment of packets when a dropping decision must be taken, which could only be beneficial to multimedia streams. By dropping less important packets (e.g. packets belonging to a B-frame or a P-frame, instead of an I-frame), an AMSP-aware router can achieve greater performance for multimedia traffic in case of congestion.

3.2. **The PDQMAMS algorithm.** The Priority-Drop Queue Management Algorithm for Multimedia Streams is essentially a queue management algorithm for a stateful AMSP-aware router that achieves fairness between AMSP-flows. Our goal in this paper was to develop a router algorithm that supports multimedia traffic inside the network. We could achieve a good result in this direction by implementing a simple queue management AMSP-aware algorithm that preferentially drops packets according to AMSP channel priorities when congestion occurs. We don't want to use for this purpose a queue scheduling algorithm since this would induce higher delays for packets inside routers; we want to use plain FIFO discipline. However, a simple AMSP-aware algorithm for queue management would have a major drawback: it would lack fairness among flows. An AMSP session (flow) will be identified by a pair of (server IP, client IP) together with a pair of (source port, destination port). Because the number of channels and the priority levels a multimedia stream should use is left to the application's choice, multiple AMSP sessions use different numbers of priority levels. Hence, if we have multiple AMSP sessions competing for bandwidth and these sessions have different numbers of priority levels (depending on each one's number of elementary streams) it is very likely that the router always selects the packet with the lowest priority from the same set of streams (belonging to a constant AMSP session). But this won't be fair to AMSP flows since the algorithm will show bias for a constant AMSP flow when a dropping decision has to be made (this flow is the AMSP flow that owns the channel with the lowest priority from all the channels of all AMSP flows). It is clear that a simple stateless AMSP-aware router will not be able to achieve fairness when drops have to be done. Consequently, our queue management algorithm must be stateful, i.e. it must maintain a state for each flow.

We consider two kinds of flows: (1) the AMSP flows and (2) one flow that contains all other non-AMSP packets (called the *other flow*), and although we have a single physical queue of packets, each flow will have its own virtual queue. As we said before, the granularity of AMSP flows will be a pair of (IP,port) binoms (one for source and one for destination). In order to keep the state maintained at the router small, we use a hash bucket with a limited number of slots for flows differentiation. When a packet arrives at the router it is first classified: if it is an AMSP packet, the packet is assigned to a slot from the hash bucket by applying the hash function on a value composed by an aggregate of source IP, destination IP, source port, destination port and transport protocol value. If the packet does not belong to an AMSP flow, it is assigned to the *other flow*. Once a packet gets assigned to an AMSP slot/flow, it is further added to the specific channel this packet belongs to in the respective AMSP flow. The number of channel is determined from the ChannelID field of the AMSP header of the packet. Each flow has a number of linked lists for every channel it has. A graphical representation of PDQMAMS's architecture is shown in Fig. 1.

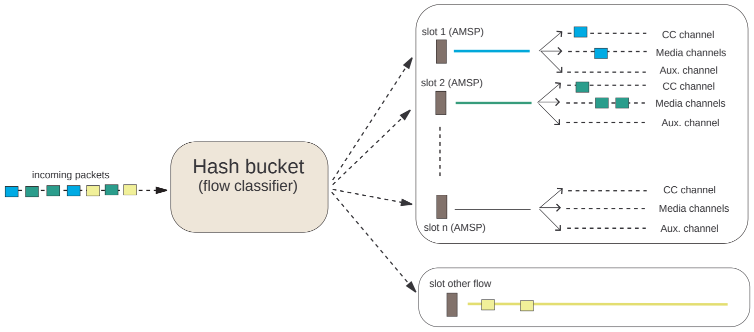


FIGURE 1. The PDQMAMS router architecture

An AMSP packet is placed on two linked lists: the global list with all packets from the queue and the list of the channel it belongs to, inside the AMSP flow. A non-AMSP packet is also placed on two linked lists: the global list and the list of packets belonging to other flows. To be noted that due to the number of slots available for the hash function, it is possible that an AMSP packet gets placed on the same slot with other AMSP packets not belonging to the same AMSP flow. To minimize the number of collisions inside a slot, the hash function is perturbed every 10 seconds. We now detail each operation PDQMAMS performs.

I. Packet enqueuing. If the queue is not full, the packet is enqueued normally. The classifier first determines if this is an AMSP packet or not. If not, the packet is added to the *other flow.* If this is an AMSP packet, based on the hash value for this packet, it is attached to an AMSP slot/flow and inside the flow is assigned to the corresponding AMSP channel. The packet is also added to the main queue. If the queue is already full when the packet arrives, one or more packets are chosen for dropping (using a sort of weighted fair dropping algorithm, see subsection III, Packet dropping) and the new packet is added to the queue the same way as shown above. This way, newer packets are favoured, as for old enqueued packets, there is a good chance they will be useless at the receiver (because their presentation time has expired).

II. Packet dequeuing. Packets are dequeued for sending using a simple FIFO strategy. This strategy is very light, easy to implement and minimizes the maximum delay of a packet in the router's queue.

III. Packet dropping When the queue is full, a dropping decision has to be made in order to accommodate a new packet. The dropping decision has to subscribe to two guidelines: it must be as fair as possible to all flows and it must protect multimedia streams. For achieving fairness, the dropping choice must be influenced by the number of packets each flow has in the queue. For the second goal, the router must always select for dropping the lowest priority packet. In order to decide which flow to drop from, all the flows (their packets) are linearly mapped on a list with the length QLen, so that all the packets of each flow are placed consecutively on this list and QLen is the number of queued packets in the router (see Fig. 2). A random integer number between 1 and QLen is generated. Let this number be k. The flow chosen for dropping is the flow that the k-th packet from the list belongs to. After the flow for dropping is chosen, we must decide which packet(s) from that flow will be dropped. If the flow is an AMSP one, we always choose for dropping the packet(s) with the lowest priority(es). If multiple packets apply, we drop the oldest one. In the case of a non-AMSP flow, we apply a tail-drop discipline. This way, a flow that has a greater number of packets in the queue, has a higher probability to be selected for dropping. This is somewhat dangerous for the *other flow* because all non-AMSP packets are assigned to this flow, thus, the length of the *other flow* can significantly surpass the length of AMPS flows, making it very vulnerable to selection when drops have to be done. This is why, we do not map all the packets from the *other flow* on the linear list, but we map only a limited number of packets from this flow (the number of all non-AMSP packets divided by a certain weight).

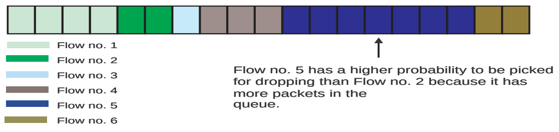

Flow no. 1
Flow no. 2
Flow no. 3
Flow no. 4
Flow no. 5
Flow no. 6
Flow no. 5 has a higher probability to be picked for dropping than Flow no. 2 because it has more packets in the queue.

FIGURE 2. Packet dropping in PDQMAMS

Note that the degree of fairness depends on the number of slots in the hash bucket. If the number of slots is great, then the number of collisions will be small and there is a bigger chance that all the packets from a slot belong to the same flow, hence, dropping fairness is increased. Conversely, if the number of slots is small, there are more collisions in the slots and packets from different flows are mixed with a higher probability. This problem is somewhat alleviated by perturbing the hash function each 10 seconds.

4. EXPERIMENTAL RESULTS

To evaluate our algorithm, we have implemented PDQMAMS as a queue discipline under the Linux kernel version 2.4.24. We used for our tests the logical network topology depicted in Fig. 3. The connection between the router R1 and router R2 was limited using the Linux traffic control framework [13] and the code corresponding to the AMSP client and AMSP server was taken from the AMSPLibrary [12]. PDQMAMS was deployed on the router R1. All the links except the R1 - R2 link have capacities higher than the bandwidths requested by our AMSP sessions, so that the only congested link is R1 - R2.

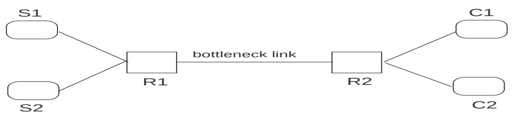


FIGURE 3. The net topology used in experiments

In the first experiment, we wanted to see how PDQMAMS supports multimedia streams by dropping lower priority packets when the queue overflows. We started one AMSP server on S1 and one AMSP client on C1 and let them run for 300 seconds. The server was sending a synthetic 512kbit/s stream with 30 fps to the client C1. The size of the frames were chosen so that I-, P- and B-frames with priority 2 fit into an AMSP packet. All other frames are larger than the MTU and are fragmented by the application. The synthetic multimedia stream was encoded using a pattern of one I-frame followed by 5 P-B-B-B-B sequences, which is common for MPEG streams. The outgoing link from the PDQMAMS router to the router R2 was limited to 64kbit/s. As you can see from Fig. 4 the percent of frames received is proportional to the frame priority. Hence, the frame type that has the highest percent of arrived frames is the I-frame which has the highest priority (zero). It is followed by P-frames which have priority 1 and then B-frames with priorities running from 2 to 6. Hence, by dropping less important packets (B-frames with priorities of 6, 5, 4 and 3), PDQMAMS increases the perceived quality of multimedia streams.

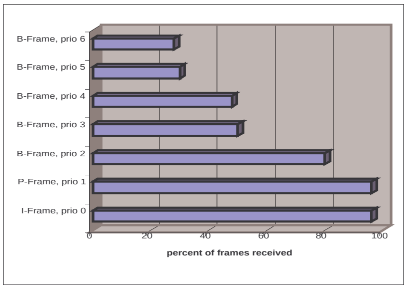
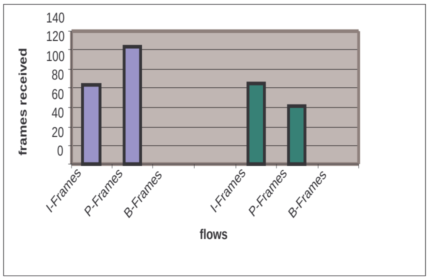
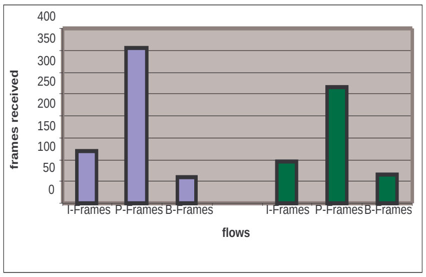


FIGURE 4. bandwidth=64kbit/s, one flow, duration=5 min.



(a) bandwidth=64kbit/s, time=2 min.



(b) bandwidth=128kbit/s, time=2 min.

FIGURE 5. Experiments with two flows (blue & green)

In the second and the third experiment we wanted to prove that PDQMAMS is fair to flows when drops are imperative. These experiments differ only in the bandwidth limitation applied to the link R1 - R2. In the experiment no. 2 this link had a capacity of 64kbit/s and in the experiment no. 3 the capacity is 128 kbit/s. Both experiments were run for 120 seconds. For both experiments two AMSP flows pass through the PDQMAMS router R1 and both flows have a bandwidth demand greater than the one provided by the network (64kbit/s for experiment 2 and 128kbit/s for experiment 3). We started two AMSP-server process on machine S1 and S2 and two clients, one on machine C1 and the other one on machine C2. The first AMSP server sends a synthetic multimedia stream to client running on C1 and the second AMSP server sends the same synthetic multimedia stream to client running on C2. The synthetic multimedia stream was the same as the one used for the first experiment. As you can see from Fig. 5(a) and Fig. 5(b) the PDQMAMS-router does a satisfactory job in achieving fairness between the two AMSP flows, while complying to the lowest priority drop scheme for supporting the quality of multimedia streams. The reason we have more P-frames received than I-frames for some AMSP flows is because in the multimedia streams, the number of I-frames is much smaller than the number of P-frames (this is compliant with the MPEG standards for achieving a better compression ratio) since the encoding pattern of the stream was one I-frame and then 5 P-B-B-B-B sequences.

5. Improvements of the algorithm

The PDQMAMS algorithm presented above, as we have seen, does a good job in supporting/protecting multimedia streams (by employing a priority-based dropping scheme) and also achieving a good degree of fairness between flows. The quality of multimedia streams is surely increased at the receiver side. However, this algorithm does not directly address the problem of congestion or, more exactly, congestion avoidance. In order to pro-actively avoid congestion inside the router while still providing fairness between flows and support for multimedia streams, the PDQMAMS algorithm needs to incorporate probabilistic dropping schemes before the queue is full like the ones employed by RED [8]. This leads us to a modified version of PDQMAMS that is still AMSP-aware and stateful, but drops are made *early*, before the queue overflows and the dropping scheme is slightly different.

We could, try on a first approach to apply a RED [8] or an Adaptive RED [14] test on the global queue. This would improve congestion avoidance and prevent buffer overflows. However, RED was mainly designed for avoiding congestion inside routers by controlling the sending rate of TCP conformant flows. So it assumes the link is utilized mostly by TCP flows (Adaptive RED relaxes to some degree this restrictions). Our improved PDQMAMS can not impose such conditions, as one of its goals is to protect multimedia streams. But a RED/Adaptive RED test can still be applied to the *other flow* where the majority of packets will belong to TCP-friendly flows. On the other hand, an AMSP flow can be subject to a RED-like test based on the average length of the flow's virtual queue and a special threshold specific for multimedia streams. This threshold should specify an upper limit for an AMSP flow's queue length, above which the packets won't be of any good at the receiver side because the delay experienced by the packet is too high (and the presentation time at the receiver had already expired before this packet would have left the queue). This threshold value can be considered as a kind of staleness threshold. If the average length of the virtual queue of an AMSP flow is greater than this staleness threshold, the lowest priority packet from this flow should be dropped.

The improved PDQMAMS algorithm outlined above will have a great chance in providing fairness for all flows, support for multimedia streams and pro-active congestion avoidance.

6. Related work

Our queue management algorithm drops lower priority packets when the router's queue becomes full. There is another approach to this prioritized treatment of packets inside a router's queue. Incoming lower priority packets can be dropped, before the queue is full, to make room for incoming higher priority packets. In this approach, an incoming lower priority packet is dropped if the number of lower priority packets from the queue is above a threshold value. This approach of managing prioritized packets inside a router's queue has the advantage of avoiding the complexity of searching the queue for a low priority packet to drop, which is done in our approach. This approach is the basis for the algorithm presented in [15]. The disadvantage of such algorithms is that they are less accurate (because they use thresholds for taking drop decisions) and they often lead to underutilisations of the queue. In [15], a version of such threshold-based algorithm which tries to alleviate this accuracy problem by using dynamic thresholds is presented. This algorithm uses not one threshold, but several different threshold values for each type of low priority frames from a multimedia stream. These thresholds are chosen in a dynamic way based on how the queue's fill level will be in the future (i.e., after several more frames are received, the *lookahead buffer*). In this algorithm, when a packet belonging to a low priority frame arrives at the router and the threshold for this kind of frame is surpassed (meaning that accepting this packet makes the lookahead buffer drop a future incoming high priority frame packet) this packet is dropped. This algorithm relies on the source sending in advance information about future incoming frames, to intermediate routers and on the fact that intermediate routers know in advance the arriving frame pattern from one second GOP.

Although our PDQMAMS does not use threshold values and drops a packet only when it's absolutely necessary (i.e., when the queue is full), thus being more accurate, complexity is added only when a packet is enqueued (like is depicted in figure 1) and, in our opinion, this complexity is comparable with the complexity of the algorithm presented in [15] when threshold values must be computed every time a packet arrives at the router. Also, the algorithm presented in [15] assumes it manages only a single multimedia flow and that available bandwidth changes on coarse-granularity time intervals, while our PDQMAMS handles several multimedia flows and achieves fairness between them.

7. Conclusions and future work

We have presented a queue management algorithm, namely Priority-Drop Queue Management Algorithm for Multimedia Streams (PDQMAMS) for helping multimedia streams inside the network. We showed that this queue management algorithm indeed provides support for multimedia streams while achieving a good degree of fairness among flows. We also sketched an improved version of PDQMAMS that tries to maintain the above qualities and also provide pro-active congestion avoidance mechanisms (mainly based on RED). As future plans we intend to implement and test this improved version of PDQMAMS under various conditions and prove its advantages.

References

[1] R. Steinmetz, K. Nahrstedt, *Multimedia: Computing, Communications and Applications*, Prentice Hall PTR, 1995.

[2] S.Floyd, K. Fall, *Promoting the Use of End-To-End congestion control in the Internet*, IEEE/ACM Transactions on Networking, August 1999.

[3] K. Ramakrishnan, S. Floyd, *A Proposal to add Explicit Congestion Notification (ECN) to IP*, RFC 2481, January 1999.

[4] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[5] Nagle, J., *Congestion Control in IP/TCP*, RFC 896, January 1984.

[6] E. Kohler, M. Handley, S. Floyd, *Datagram Congestion Control Protocol (DCCP)*, Internet Draft, February 2004, http://www.icir.org/kohler/dcp/draft-ietf-dccp-spec-06.txt.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, July 2003.

[8] S. Floyd, V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 397-413, August 1993.

[9] P. E. McKenney, *Stochastic Fairness Queuing*, Internetworking: Research and Experience, Vol. 2, pp. 113-131, 1991.

[10] A. Demers, S. Keshaw, S. Shenker, *Analysis and Simulation of a Fair Queuing Algorithm*, Journal of Internetworking: Research and Experience, 1, 1990, pp. 3-26.

[11] MPEG Standards, http://www.chiariglione.org/mpeg/index.htm.

[12] M. Ohlenroth, *Network-based Adaptation of Multimedia Content*, PhD thesis, Klagenfurt University, Austria, September 2003.

[13] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans and P. Larroy, *Linux Advanced Routing and Traffic Control*, August 2003, available at http://lartc.org/howto.

[14] S. Floyd, R. Gummadi, S. Shenker, *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management*, august, 2001.

[15] A. Awad, M.W. McKinnon and R. Sivakumar, *MPFD: A Lookahead Based Buffer Management Scheme for MPEG-2 Video traffic*, Proceedings of the 8th IEEE International Symposium on Computers and Communication, 2003, (ISCC 2003).

Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca
E-mail address: {forest, florin, bufny, claudiu}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

BROAD PHONETIC CLASSES EXPRESSING SPEAKER INDIVIDUALITY

MARGIT ANTAL, GAVRIL TODEREAN

Abstract. Vector quantisation and Gaussian mixture modelling methods are very popular methods for automatic speaker identification. First we give a concise overview of these methods, then present some measurements comparing them on behalf of the TIMIT corpus. The aim of this paper is to study the influence of the speech material on performances of such methods. For this purpose pure phonetic speaker models were created containing speech data from a single broad phonetic class. The speaker discriminative property of these pure phonetic speaker models had been investigated. Among the broad phonetic classes nasals and vowels were found to be particularly speaker specific.

Key Words: Speaker Identification, Gaussian Mixture Models, Pure phonetic speaker models

1. Introduction

A variety of signals and measurements have been proposed and investigated for use in biometric recognition systems. Among the most popular measurements are fingerprint, face and voice. There are two main reasons for using voice instead of other measurements. First, there is a well-developed infrastructure for speech signal transmission which can be accessed almost everywhere using a cell phone. Second, speech is the most natural way of communication, therefore is not intrusive for users to provide speech sample for authentication.

Speaker recognition is the process of recognising the speaker on the basis of information obtained from speech waves. Speaker recognition can be divided into speaker identification and speaker verification. While speaker identification is a classification problem performed on a closed set of speakers, speaker verification is a binary decision, determining whether an unknown voice is from a particular enrolled speaker. If the speaker is recognised based on unconstrained speech, the system is called text-independent. However, text constrains can greatly improve the accuracy of a system. A great overview of speaker recognition systems can be found in [10].

Received by the editors: July 28, 2005.

2000 *Mathematics Subject Classification.* 68T10, 62H30.

1998 *CR Categories and Descriptors.* I.5.2 [**Pattern Recognition**]: Design Methodology– *Classifier Design and Evaluation*; I.5.4 [**Computer Systems Organization**]: Special-purpose and Application-based Systems – *Signal Processing Systems* .

The special recognition task addressed in commercial systems is that of verification rather than identification. In spite of that, for this project, we chose to confine our experiment to the task of closed-set identification rather than speaker verification. The motivation for doing so was to measure the classification capability of the system without having to consider the effect of different background model normalisation schemes required for the verification task.

The majority of research papers focuses on feature extraction and selection methods or classifier combinations for obtaining higher identification rates rather than on analysing the content of speaker models. Flanagan's group in [9] selectively used the speech spectrum for speaker identification and found that the higher portion of the speech spectrum contains more reliable idiosyncratic information on the speaker.

The aim of this paper is to investigate the discriminative properties of several broad phonetic classes in this special pattern classification problem, the speaker identification. Previous works in this field using other speaker models were done in [8, 4]. While in [4] a well-known French speech database was used, in [8] the authors worked on a private English database. No similar results were reported on TIMIT database.

Section 2 briefly presents the speaker modelling techniques. In Section 3 we review the main characteristics of the broad phonetic classes. Section 4 presents experiments on speaker identification using different types of features and models trained with broad phonetic classes. Section 5 presents statistical analysis of the content of a VQ based speaker model. Finally, in Section 6 we discuss the results and draw the main conclusions of our paper.

2. Speaker models

Over the past several years, Gaussian mixture models have become the dominant approach for modelling in text-independent speaker recognition applications [19]. However, in special cases, simpler speaker models could perform similarly well or even better. One simpler model is the vector quantisation (VQ) model, which was investigated in several papers [5, 20]. Other papers compare the GMM and VQ models drawing conclusions based on measurements on different speech databases [21, 16]. Recently the two methods have been successfully combined resulting in the VQGMM method [9].

According to Jain et al. [11], CA (Clustering Algorithm) is the organisation of a collection of patterns, represented as multidimensional feature vectors, into clusters based on similarity. Patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. Vector Quantisation is not so much interested in finding the clusters, but in representing the data by a reduced number of elements that approximate the original data set as well as possible. We can say that in many cases CA and VQ are practically equivalent, grouping the data into a certain number of groups, so that an error function is minimised.

2.1. **Vector Quantisation - VQ.**

The objective of VQ is the representation of a set of feature vectors $X = \{x_1, x_2, \ldots, x_N\} \subseteq R^D$ by a set $Y = \{y_1, y_2, \ldots, y_M\}$, of M reference vectors in R^D. Y is called codebook and its elements codewords. VQ can be represented as a function $q : X \to Y$. The function q permits us to obtain a partition S of X constituted by M subsets S_i, where each cell S_i has the form

(1) $$S_i = \{x \in X \quad : \quad q(x) = y_i\}, \quad i = 1, \ldots, M.$$

We measure the goodness of partitioning by the means of quantisation error(MQE), which can be defined as follows

(2) $$MQE = \frac{1}{M}\sum_{i=1}^{M} D_i, \; where \; D_i = \sum_{x_j \in S_j} d(x_j, y_i)$$

where d is the Euclidean distance defined in R^D.

VQ can be done using different quantisation algorithms. The simplest one is the LBG algorithm [15], which was recently enhanced into ELBG in [17]. All variants of LVQ introduced by [14] can be applied equally well. All the clustering algorithms developed by Artificial Intelligence researchers might work as well. A comparison of several clustering algorithms used in speaker identification was done in [13, 2].

2.2. **Gaussian Mixture Models - GMM.**

Finite mixture is a flexible and powerful probabilistic tool. Mixtures can also be seen as a class of models that are able to represent arbitrarily complex probability density functions.

For a $D-$dimensional feature vector, x, the mixture density used for the likelihood function is defined as

(3) $$p(x|\lambda) = \sum_{i=1}^{M} w_i p_i(x)$$

The density is a weighted linear combination of M unimodal Gaussian densities, $p_i(x)$, each parameterised by a mean vector μ_i, and a covariance matrix, Σ_i

(4) $$p_i(x) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} e^{-\frac{(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}{2}}$$

The mixture weights, w_i, satisfy the constraint $\sum_{i=1}^{M} w_i = 1$. A GMM model can be denoted as

(5) $$\lambda = \{w_i, \mu_i, \Sigma_i\}, \quad i = 1, \ldots, M.$$

Given a collection of training vectors, the expectation-maximisation (EM) [7] algorithm can be used to estimate the model parameters. This algorithm iteratively refines the GMM parameters in order to monotonically increase the likelihood of the estimated model for the observed feature vectors.

A new parameter estimation method was proposed in paper [9]. The paper proposes to cluster the whole acoustic space into several subspaces. Within a subspace, the feature vectors are relatively more homogeneous. Each subspace is then characterised by a number of Gaussian mixture models whose parameters are

Table 1. Corpus division

Dataset	Utterances	Length
training	2 SA, 3 SX, 3 SI	24.5s
test	2 SX	6.06s

determined using only those relevant acoustic features belonging to the subspace. This means that feature vectors far from the subspace are not used to estimate model parameters for that subspace.

3. Phonetic classes

The basic theoretical unit for describing how speech conveys linguistic meanings is called a phoneme. Each phoneme can be considered to be a code that consists of a unique set of articulatory gestures. These articulatory gestures include the type and location of sound excitation, as well as the position of movement of the vocal tract articulators. There are some phonetic alphabets in use. European phoneticians developed the International Phonetic Alphabet (IPA), which is appropriate for handwritten transcription but its main drawback is that it cannot be typed on a conventional typewriter or a computer keyboard. Therefore, a more recent phonetic alphabet was developed by the United States Advanced Research Projects Agency (ARPA), and is accordingly called ARPAbet.

There are a variety of methods for classifying phonemes. Phonemes can be grouped based on properties related to the time waveform or frequency characteristics. A phoneme is continuant if the speech sound is produced by a steady-state vocal-tract configuration. A phoneme is non continuant if a change in the vocal-tract configuration is required during production of the speech sound. Vowels, fricatives, affricates, and nasals are all continuant sounds. Diphthongs, liquids, glides, and stops all require a vocal-tract reconfiguration during production. An exhaustive study of these classes can be found in the following books [18, 6]. In our study liquids and stops are grouped together forming the semivowels group.

Experiments were performed on the TIMIT corpus, which is phonetically segmented and annotated using the ARPAbet symbols. For broad phonetic classes we used those recommended in TIMIT corpus documentation, which are the following: Vowels, Semivowels, Nasals, Stops, Fricatives, Affricates, Silence+Closures. The speech corpus consists of 10 spoken utterances from 630 speakers covering the 8 major dialect regions of the United States. Table 1 shows the speech corpus division in training and test utterances and table 2 shows the average length of speech material for each broad phonetic class. There is no point of making speaker models from silence and we could not use the affricates, their average length were not enough to train the models.

4. Speaker Identification Experiments

All the experiments were conducted on the TIMIT speech corpus, using all 630 speakers for speaker identification.

TABLE 2. Broad phonetic classes and their training and test length

Phonetic class	Phonemes	Training length	Test length
Vowels	iy,ih,eh,ey,ae,aa,aw ay,ah,ao, oy,ow,uh,uw ux,er,ax,ix,axr,ax-h	9.66s	2.27s
Semivowels	l,r,w,y,hh,hv,el	2.38s	0.47s
Nasals	m,n,ng,em,en,eng,nx	1.34s	0.38s
Fricatives	s,sh,z,zh,f,th,v,dh	3.28s	0.95s
Stops	b,d,g,p,t,k,dx,q	1.43s	0.35s
Affricates	jh,ch	0.20s	0.09s
Silence+Closures	pau,epi,h# bcl,dcl,gcl,pcl,tck,kcl,dcl,tcl	6.28s	1.55s

Before segmenting the signal into frames, a filter was applied to enhance the high frequencies of the spectrum. We used the following filter:

$$x_p(t) = x(t) - a * x(t-1)$$

where $a = 0.97$.

The analysis of speech signal was done locally by the application of a window whose duration in time is shorter than the whole signal. This window is first applied to the beginning of the signal, then moved further and so on until the end of the signal is reached. For the length of the window we used 32ms with 22ms of overlapping between consecutive frames. Each frame was multiplied by a Hamming window in order to taper the original signal on the sides and thus reduce the side effect. After these steps we extracted cepstral parameters from each frame. In the following experiments we used two types of cepstral features, MFCC and LPCC. The detailed description of these features can be found in [3]

4.1. **VQ and GMM comparison.** For these speaker identification experiments we used LPCC features, which performed slightly better than the MFCC ones. Both VQ and GMM models were trained with 32 components. For VQ we used the LBG algorithm and the GMM models were initialized by the mean vectors provided by the LBG algorithm. The weights were set to be equal. We used diagonal covariance matrices initialised with the identity matrix. The standard ML estimation of the parameters was used with 10 iterations.

The results are summarised in Table 3. We used the training-test division presented in Table 1.

Similar results were reported for the case of GMM in [19] and for VQ in [12] , all measured on the same speech corpus and using cepstral features.

4.2. **Phonetic pure GMM.** The aim of these experiments is to determine the speaker discriminative phonetic broad classes. Table 4 summarises the identification rates obtained for the phonetic broad classes. The amount of data used for training and test is presented in Table 2. In these experiments we used 12 MFCC parameters. The number of mixture's density components were selected

Table 3. Speaker identification results using all the 630 speakers from TIMIT

Features	VQ-LBG	GMM
LPCC-12	97.40%	98.26%
LPCC-16	99.05%	99.05%
LPCC-20	99.30%	99.70%
LPCC-24	100.%	99.85%

Table 4. Speaker identification rates for 630 speakers using pure phonetic GMMs

Phoneme class	Training	Test	Mixtures	Id. rate
Vowels	9.65s	2.27s	8	95.39%
Nasals	1.34s	0.38s	1	70.31%
Fricatives	3.27s	0.95s	4	44.60%
Semivowels	2.38s	0.47s	4	41.74%
Stops	1.43s	0.35s	4	10.47%
All	24.55s	6.06s	32	96.20%

carefully, running several times the classification for different number of mixtures and selecting the one, which gives the best result.

The result obtained for the vowels is amazing. This means that using homogeneous data, which represents only 40% of the whole training data, we could almost reach the performance of the models using all training data. Another impressive result was produced by the nasals group, which represents approximately 5-6% of the whole speech data.

The results summarised in Table 4 are representative for the TIMIT database but are not comparable due to the variety of training and test time. For a correct ranking of the discriminative effects of the broad phonetic classes on speaker identification, we limited all training data to 1.5s and the test data to 0.5s for every speaker. For every classification we used a GMM with two density components. Table 5 ranks the identification rates obtained in similar training and test conditions for broad phoneme classes. We included for comparison a similar test using all phonetic classes, 1.5s training and 0.5s test data. We can see that limiting the training and test material seriously affected vowels and fricatives.

5. Phonetic content of speaker models

In this section we are going to analyse the phonetic content of VQ based speaker model. This model consists of a set of clusters. Eeach cluster is represented by its centroid, which is chosen as a prototype of its cluster. Our goal is to verify how well the broad phonetic classes are separated in this type of speaker model. We selected several speakers and made a statistical analysis of the content of their models. Our analysis has the following steps:

TABLE 5. Identification rates for 630 speakers using in average 1.5s training and 0.5s test data and a GMM model with 2 density components

Phonetic class	Id. rate
Nasals	64.92%
Vowels	20.31%
Semivowels	19.73%
Fricatives	11.42%
Stops	10.15%
All	12.38%

(1) First, we obtained the feature vectors from all audio data belonging to the selected speaker. We excluded the feature vectors for silence, because these vectors do not contain speaker specific data. Let us denote this set by $X = \{x_1,\ x_2,\ \ldots, x_T\}$, where $x_i \in R^D$and D is the dimensionality of the feature space.
(2) In the second step we labeled each feature vector with its broad phonetic class label. Let us denote by $Y = \{(x_i,\ f_i) \mid i = 1, 2, \ldots T\}$ the resulted set, where $f_i \in F$. $F = \{A,\ F,\ S,\ W,\ V,\ N\}$ is the set of broad phonetic class labels. Label A denotes the affricates, F is for fricatives, S is the symbol for stop phonemes, W and V denote the semivowels and vowels and N stands for nasals.
(3) Using these labeled feature vectors we applied the VQ algorithm and obtained the M clusters, whose content we analysed statistically.

Figure 1 shows the distribution of broad phonetic classes in the $M = \{2,\ 3,\ 4,\ 5,\ 6\}$ clusters created by the clustering algorithm. We repeated the clustering using up to 6 clusters in order to be able to capture the broad phonetic classes separation tendency. We stopped at 6, because there are altogether 6 broad phonetic classes.

The left top figure shows that if we use only two clusters, the first one will be populated by affricates, fricatives and stops and the second one by vowels, semivowels and nasals. This clustering tendency is explainable by the acoustical similarities between these broad phonetic groups. The affricates group is the least numerous one, and these phonemes are always situated in the same cluster together with the vast majority of fricatives and a representative part of the stop broad phonetic group.

Analysing the last figure, in which we used exactly six clusters, we can see that the vast majority of affricates, fricatives and nasals are concentrated in one cluster and only a small amount are spread among other clusters. Because we used all frames from phonemes, some of these frames are situated at the boundaries of phonemes, not being very representative for any broad phonetic class.

This experiment shows that a speaker model does not separate perfectly the broad phonetic classes. It is also true that every broad phonetic group has its own specific clusters. The bigger the acoustic variety inside a broad phonetic group the more clusters are spread among the feature vectors belonging to these groups.

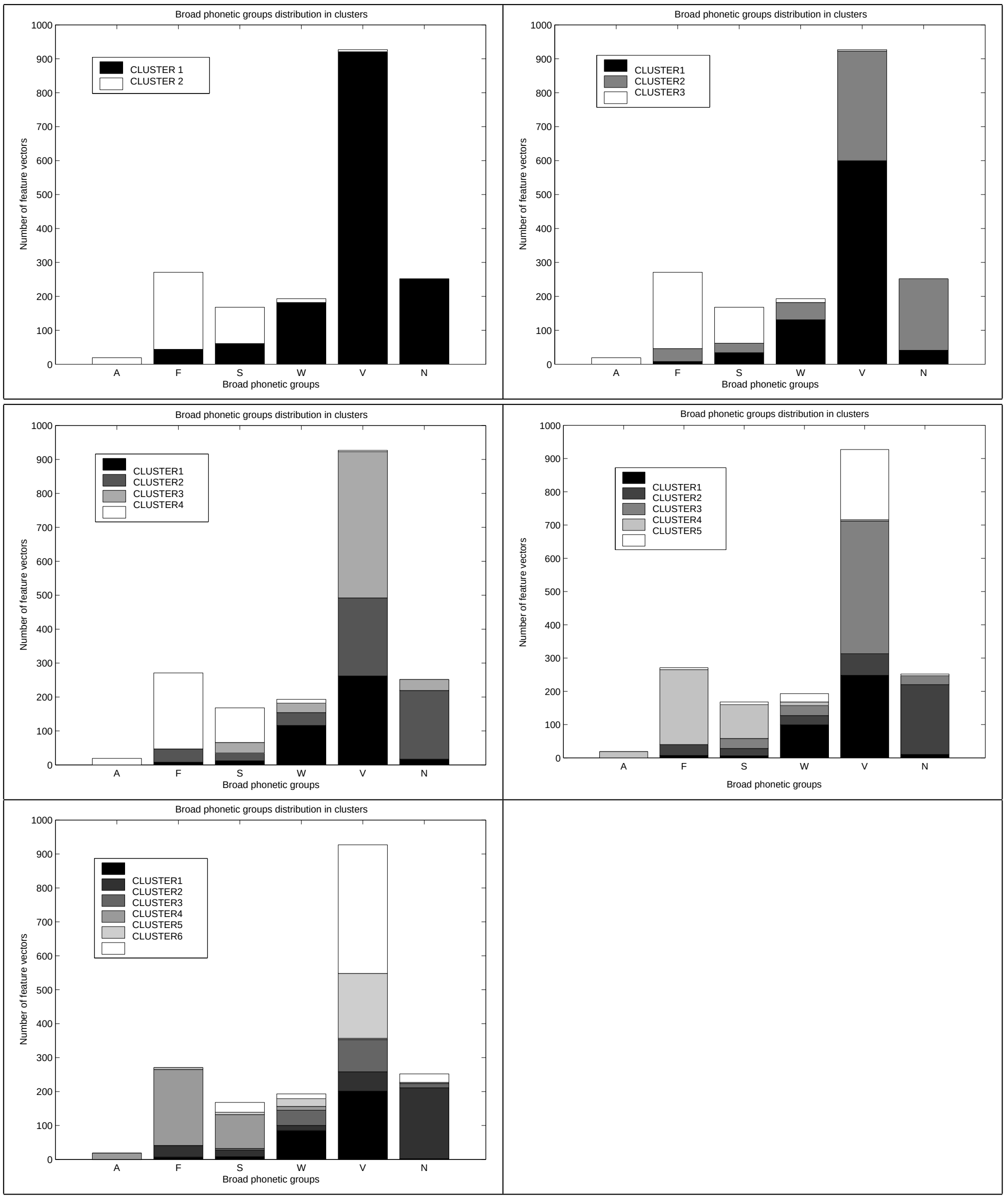


FIGURE 1. Broad phonetic classes distribution in clusters

Broad phonetic class	Variance
Affricates	23.22
Fricatives	47.35
Stops	37.85
Semivowels	46.76
Vowels	41.33
Nasals	20.53

Table 6. Variances of broad phonetic classes

The acoustic variety of a broad phonetic group can be characterized by the variance of the group. The higher this variance the more powerful the separating tendency of the group. We computed for each speaker model the variances of the broad phonetic classes. For this experiment we used the training part of the TIMIT speech corpus (462 speakers). Table 6 shows the average values of these variances. The higher the speaker discriminating ability of a broad phonetic group, the lower the variance of this class in a speaker model.

6. Conclusions

The main purpose of this paper was to compare the relative speaker discriminating properties of broad phonetic classes. For this purpose pure phonetic speaker models were created. We found that the pure phonetic speaker models using exclusively vowels, almost reached the performance of models using the whole speech data from a speaker. We should mention that the vowels represent 40% of the whole corpus. We also found that when pure phonetic speaker models were trained using the same amount of training data, the nasals produced the best identification rate. We can conclude that for a very good speaker model one should use speech materials which contain as much nasals as possible. Another conclusion is that the phonetic content of the training speech material is more important than its quantity. We have also studied the distribution of broad phonetic classes in the components of speaker models. We showed that the clusters of a speaker model are not phonetically pure. However, every broad phonetic class has its specific components. We showed experimentally that nasals have the best speaker discriminating ability and also the lowest variance per speaker.

Part of this work was published in [1].

References

[1] Antal, M., Toderean, G., Speaker Recognition and Broad Phonetic Groups, *Proceedings of the 3rd IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, February 15-17, Innsbruck, Austria, 2006, 155-159.

[2] Antal, M., A Comparison of Parametric Clustering Techniques used in Speaker Identification, *Proc. of the 1st International Conference on Intelligent Knowledge Systems*, Assos, Turkey, 2004, 19-25.

[3] Bimbot, F., Bonastre, J-F, Fredouille, C., Gravier, G., Margin-Chagnolleau, I., Meignier, S., Merlin, T., Ortega-Gracia, J., Petrovska-Delacretaz, D., Reynolds, D. A., A Tutorial on Text-Independent Speaker Verification, *EURASIP Journal on Applied Signal Processing*, 4, 2004, 430-451.

[4] Chagnolleau,I.M., Bonastre, J-F., Bimbot, F., Effect of utterance duration and phonetic content on speaker identification using second order statistical methods, *Proc. Eurospeech*, Madrid, Spain, 1995, 337-340.

[5] Campbell, J. P., Speaker Recognition: A Tutorial, *Proc. of the IEEE*, vol. 85(9), 1997, 1437-1462.

[6] Deller, J.R., Hansen, J. H.L., Proakis, J. G., *Discrete-Time Processing of Speech Signal*(IEEE Press, 2000).

[7] Dempster, A., Laird, N., Rubin, D., Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society*, 39(1), 1977, 1-38.

[8] Eatock, J.P., Mason, J. S., A quantitative assessment of the relative speaker discriminating properties of phonemes, *Proc. International Conference on Acoustics, Speech, and Signal Processing*, Adelaide, Australia, 1994, 333-336.

[9] Lin, Q., Jan, E-E.,Che, C., Yuk, D-S, Flanagan, J., Selective use of the speech spectrum and a VQGMM method for speaker identification, *Proc. 4th International Conference on Spoken Language Processing*, Atlanta, USA, 1996,1321-1324.

[10] Furui, S., An overview of speaker recognition technology, in C-H. Lee, F.K. Soong, K.K. Paliwal (Eds.) *Automatic Speech and Speaker Recognition, Advanced Topics*, (Kluwer Academic Publisher, 1996) 31-56.

[11] Jain, A.K., Murty, M.N., Flynn, P.J., Data clustering: A review, *ACM Computing Surveys*, 31(3), 1999, 264-323.

[12] Kinnunen, T., Karpov, E., Franti, P., Real-Time Speaker Identification, *Proc. of 8th Int. Conference on Spoken Language Processing*, 2004, 1805-1808.

[13] Kinnunen, T., Kilpelainen, T., Franti, O., Comparison of clustering algorithms in speaker identification, *Proc. 4th IASTED International Conference on Signal Processing and Communications*, Marbella, Spain, 2000, 222-227.

[14] Kohonen, T., *Self Organizing Maps* (Berlin, Springer, 2001).

[15] Linde, Y., Buzo, A., Gray, R. M., An algorithm for vector quantizer design, *IEEE Transactions on Communications*, 28(1), 1980, 84-94.

[16] Matsui, T., Furui, S., Comparison of text-independent speaker recognition methods using VQ-distortion and discrete/continuous HMMs, *Proc. ICASSP*, San-Francisco, California, 1992, 157-160.

[17] Patene, G., Russo, M., The enhanced LBG algorithm, *Neural Networks*, 14(9), 2001, 1219-1237.

[18] Rabiner,L.R., Juang, B. H., *Fundamentals of speech Recognition*(Englewood Cliffs NJ: Prentice-Hall, 1993).

[19] Reynolds, D.A., Speaker identification and verification using Gaussian mixture speaker models, *Speech Communications* 17(1-2), 1995, 91-108.

[20] Soong,R. K., Rosenberg, A. E. , Juang, B. H., Rabiner, L. R., A Vector Quantization Approach To Speaker Recognition, *AT&T Technical Journal*, 66 (6), pp. 14-26, 1987.

[21] Stapert, R., Mason, J.S., Speaker Recognition and the Acoustic Speech Space, *Proc. Odyssey Speaker Recognition Workshop*, Crete, Greece, 2001, 195-199.

Sapientia - Hungarian University of Transylvania,Faculty of Technological and Human Sciences, 540053 Tg.-Mures, Romania, Technical University of Cluj-Napoca, Faculty of Electronics and Telecommunications, Romania

E-mail address: manyi@ms.sapientia.ro, toderean@com.utcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

PREFIX-FREE LANGUAGES, SIMPLE GRAMMARS REPRESENTING A GROUP ELEMENT, LANGUAGES OF PARTIAL ORDER IN A GROUP

KRASSIMIR D. TARKALANOV

Abstract. We show each word in the Kleene closure of a prefix-free language over an arbitrary alphabet has only one presentation as a concatenation of its words. It follows this language is the largest prefix-free one in the closure and both of them are simultaneously recursive or not recursive. We note if a complete simple grammar generates words only from a prefix-free language, the generated language exhausts it entirely. Such particular results only for word- and reduced word problem languages of a group can be found in [1, 4]. Using appropriate parts of the repeated in [4] construction from [1] we construct entire simple grammars whose terminal set is the monoid generating set of a group. The start symbol can be indexed by any element of the group and then the corresponding grammar will generate only representatives of this element. If all of them contain in its prefix-free part, their set exhausts this part according to the note above. Following [4] the reduced word problem language must be a such part of the word problem language for a group with finite irreducible word problem and the simple grammar there. The answer to the final question 5.3. [4] is absolutely analogous and simply follows the proof from [1]. We give necessary and sufficient conditions which a language must satisfy together with these [4] for a word problem language in order the first one to assign a partial order in the group of the second one.

1. Necessary Preliminary Concepts

We will use concepts from [1,4] in sections 2 and 3, and in addition from [2] in section 4. Our supplement is their concrete setting in an order, their connecting, and some formulations. A definition 1.1. for entire simple grammar is added below. We have corrected here the principle for a right inverse element as it is in [4] with the principle for a left inverse element because the requirement for a right one leads to another situation [3]. We note a generalization of the used in

Received by the editors: April 1, 2006.

2000 *Mathematics Subject Classification.* 03D03, 06F15, 20F10, 20M05, 68Q45, 68Q70, 68R15.

1998 *CR Categories and Descriptors.* **F.4.2** [**Grammars and Other Rewriting Systems**]: Decision problems, Grammar types, Thue systems; **F.4.3** [**Formal Languages**]: Algebraic language theory, Classes defined by grammars or automata, Operations on languages; **G.2.1** [**Combinatorics**]: Combinatorial algorithms.

[4] syntactic congruence was used by us much earlier in [5], [6] for demonstration a pure subsemigroup of a group can not be covered by a regular language and for obtaining finite homomorphic images of some semigroups.

For Reading Section 2. For any set Σ let Σ^* denotes the set of all *finite words* in the elements of Σ, i.e. Σ^* is the set of all finite strings of these elements (Λ is the empty word). The number of the symbols in a such word is *its length.* The expression $s \equiv t$ means s and t are *identical* as strings of symbols. The word st is *a concatenation* of the word t after the word s. Any subset of Σ^* is *a language* in it. If L is a language, then *its Kleene closure* L^* in Σ^* is the language which consists exactly of all finite concatenations of the words from L. If $t \equiv us$, then u is said to be *a prefix* of t. It is *a proper prefix* if it is nonempty and ends before the end of t. Given a language L *the notation* $MIN(L)$ denotes the set of all words in L each one of which has no proper prefix in L. *A language* L is said to be *prefix-free* if $MIN(L) = L$. $MIN(L)$ is a prefix-free language. Analogically, about *suffix-free languages.*

A grammar is a four-tuple $\Gamma = (N, \Sigma, P, S)$ where N is the set of its *non-terminal symbols*, Σ is its set of *terminal symbols* all different from non-terminal ones, S is a non-terminal symbol of N called *start symbol*, and P is its *set of productions.* Each production has the form $\alpha \to \beta$ in which α and β are words from $(N \cup \Sigma)^*$ and α contains at least one non-terminal symbol. The word $\alpha_1\beta\alpha_2$ is *directly derived* from the word $\alpha_1\alpha\alpha_2$ by this production. Each sequence of direct derivations gives *a derivation* of its last word from the first one. *The language* $L(\Gamma)$ *generated by the grammar* $\Gamma = (N, \Sigma, P, S)$ is the set of all words over the terminal alphabet Σ which can be derived from the start symbol S. A grammar $\Gamma = (N, \Sigma, P, S)$ is *context-free* if each production has the form $A \to \beta$ where A is a non-terminal symbol. The generated by context-free grammars languages are *context-free languages.* Every such language can be generated by a grammar in Greibach normal form. *A context-free grammar is in Greibach normal form* if it contains no non-terminal symbols which do not participate in a derivation of some terminal word and if each production has one of the forms

$A \to aB_1B_2...B_n$,

$A \to a$, or

$S \to \Lambda$.

Here A is a non-terminal symbol, each B_i too and other than S, and a is a terminal symbol. A grammar in Greibach normal form is *simple* if for each non-terminal symbol and each terminal symbol no more than one production of the indicated form is allowed, i.e. if $A \to a\alpha$ and $A \to a\beta$ are productions, then $\alpha \equiv \beta$ and if $S \to \Lambda$ is a production, then it is only one. *A language is simple* if it can be generated by a simple grammar. In [1], lemma 2. *it is proven a simple language is prefix-free.* The author uses only *leftmost derivations* at each step of which the leftmost participation of a non-terminal symbol is separated: if $\alpha_1 A\alpha_2 \to \alpha_1\beta\alpha_2$ is a step in a leftmost derivation made by using the production $A \to \beta$, then α_1 is a terminal word, i.e. it contains only terminal symbols. The author shows *every word in a context-free language has a leftmost derivation* which can be obtained simply by changing the order in which productions are used in an arbitrary derivation.

Definition 1.1. *A simple grammar is entire if for each non-terminal and each terminal symbols it contains a production of the indicated above forms.* Then it contains only one production of the indicated forms for each pair of non-terminal and terminal symbols. Some of the above concepts will be used for reading the next section.

Other Concepts for Reading Section 3. In this section Σ will be *a double alphabet* $\Sigma = \{x_1, x_2, \ldots, x_n; \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$. If a_i is x_i, then $\overline{a_i}$ is $\overline{x_i}$; if a_i is $\overline{x_i}$, then $\overline{a_i}$ is x_i $(i = 1, 2, \ldots, n)$. For each word $u \equiv a_{i_1} a_{i_2} \ldots\ a_{i_k}$ in Σ its *inverse word* $\overline{u}$ is $\overline{u} \equiv \overline{a_{i_k}} \ldots \overline{a_{i_2}}\ \overline{a_{i_1}}$ in which the symbols are inverse and set in the inverse order. Let G be a finitely generated group with a monoid generating finite set Σ (or with a group generating set $\Sigma^+ = \{x_1, x_2, \ldots, xn\}$ or $\Sigma^- = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$) and with a finite set of defining relators. Each element of the group G is presented by words from Σ^*. *The natural correspondence* φ *depicts* each generator a_i into the element $\varphi(a_i)$ of G which contains a_i. *This correspondence is extended inductively* for each word and so we receive *the natural homomorpphism* φ of Σ^* over G. *The word problem language of this group* is the full prototype $\varphi^{-1}(\imath)$ of the unite element $\imath$ in G. *The reduced word problem language* is its part of all words each one of which does not have a proper prefix in it. If we denote the word problem language with $\mathcal{E}$ $(\mathcal{E} = \varphi^{-1}(\imath))$ and the reduced word problem language with $\mathcal{R}$, then $\mathcal{R} = MIN(\mathcal{E})$ according to the above notation of the function MIN. *The irreducible word problem language* is the set of all words from the (reduced) word problem language which have no proper subwords from the word problem language. It is not especially necessary to note these languages are shortly correspondingly named word problem, reduced word problem, and irreducible word problem only in [1, 4]. We prefer the terms with an added word "language" because they are language interpretations related to the word problem in the theory of groups.

Additional Concepts for Reading Section 4. *A group is partially ordered* if there is a partial order $\geq$ in it which is concerted with the group operation. That means if $a \geq b$ in the group, then $ax \geq bx$ and $xa \geq xb$ in it. The multiplication preserves the strong inequequality. The set of all strongly positive elements (i.e. of the elements which are strongly bigger than the unite element) is called *a strongly positive cone of the partially ordered group* and it is *a pure subsemigroup* of this group. That means it is an invariant subsemigroup which does not contain the unite element. Conversely, each pure subsemigroup assigns a partial order in the corresponding group.

2. Kleene Closures of Prefix-Free Languages in an Arbitrary Alphabet and Entire Simple Grammars

In this section the alphabet Σ is arbitrary. The general properties of the prefix-free languages below are induced by the properties of word problem and reduced word problem languages.

Proposition 2.1. *Let R be a prefix-free language in Σ^* and R^* is its Kleene closure. Then*

(1) Each word from R^* has only one presentation as a concatenation of words from R;

(2) $MIN(R^*) = R$;

This property can be expressed in an equivalent form:

(2') R is the largest prefix-free language in R^* (i.e. there exists no a prefix-free extension of R in R^*). R^* by itself is not prefix-free of course.

Proof. (1) If the word w from R^* is empty ($w \equiv \Lambda$), there is nothing to prove. Let w is a nonempty word from R^* and it has two presentations as concatenations of nonempty words from R:

$w \equiv v_1 v_2 ... v_k$, where $v_1, v_2, ..., v_k \in R$, and

$w \equiv w_1 w_2 ... w_l$, where $w_1, w_2, ..., w_l \in R$.

We have to prove $k = l$ and $v_1 \equiv w_1$, $v_2 \equiv w_2$, ..., $v_k \equiv w_k$. The proof is inductive with respect to the sum $k + l$ of the numbers of the factors in these presentations. Its minimal value is 2. Then $w \equiv v_1 \equiv w_1$ and the statement is obvious. Let it be true for all natural numbers whose sum is less than $k+l$. The first factors v_1 and w_1 from the prefix-free language R in the indicated presentations of w coincide because each one of them can't be a proper prefix of the other one. Therefore we have the presentations $w' \equiv v_2 ... v_k \equiv w_2 ... w_l$ of the remaining part w' of the word w after v_1 ($v_1 \equiv w_1$). The sum of the numbers of the factors in these presentations is $k + l - 2$. According to the inductive conjecture we receive $k - 1 = l - 1$ and $v_2 \equiv w_2$, ..., $v_k \equiv w_k$ for the all next factors.

(2) $MIN(R^*)$ is the subset of all words in R^* which one of which has no proper prefix in it (R^*). Each word in R has no proper prefix in itself. It follows from this each such word can not have a proper prefix in R^* because every word in R^* is a concatenation of words from R and, then, it would follow its first factor from R would be a proper prefix of a word in R. This is impossible because R is a prefix-free language. Therefore $R \sqsubseteq MIN(R^*)$. Conversely, each word in $MIN(R^*)$ can not have more than one factor in its presentation as a concatenation of R-words because, in the opposite case, its first factor would be a proper prefix of its in R^*. Therefore $MIN(R^*) \sqsubseteq R$ and $MIN(R^*) = R$.

Our idea for the Kleene closure of an arbitrary prefix-free language comes from [1,4] where the authors prove statements for word problem and reduced word problem languages in a double alphabet only:

Proposition 3.1. [4] *The word problem of a group with respect to a monoid generating set is the Kleene closure of its reduced word problem with respect to that generating set.*

Proposition 3.2. [4] *If W is the word problem of a group with respect to a monoid generating set and R is the reduced word problem with respect to this set, then $R = MIN(W) \cap X^+$.*

Lemma 4. [1] *If π is a finitely generated group presentation and L is prefix-free language such that $WP_0(\pi) \sqsubseteq L \subset WP(\pi)$, then $L = WP_0(\pi)$.* In the notations of the author there $WP(\pi)$ is the word problem (language) of π and $WP_0(\pi)$ is its reduced word problem (language).

The property (1) is nowhere else indicated and obviously it is very important at all due to the universal only one presentation. It has an application for the proof of the simultaneous recursiveness below. Property (2) (indicated in [4], proposition 3.2, for reduced word problem and word problem languages only) is important due to an unification of different requirements to the prefixes in the definitions and, in addition, then it is not necessary to show its equivalent form (2') as a property which is separated from the presentation, as this is done above in [1], lemma 4.

Corollary 2.2. *If R is a prefix-free language in Σ^* and R^* is its Kleene closure, then both of them are simultaneously recursive or not recursive.*

The proof is practically the same as in [4], theorem 3.5., where the formulation and the proof are again for reduced word problem and word problem languages in a double alphabet only, but using the previous proposition of ours here and without passing through the recursive enumerating of R.

Let R be recursive in Σ^*, i.e. there exists an effective procedure $\mathcal{A}$ for recognizing whether a word is from R or not. We will show then there exists an effective procedure $\mathcal{B}$ for recognizing belonging of any word to R^* without passing through the recursive enumerating of R as it is in [4]. Let w is an arbitrary word from Σ^*. We apply the algorithm $\mathcal{A}$ to its beginning. If $\mathcal{A}$ stops at some prefix of w showing this prefix is from R, we denote it with v_1, i.e. $w \equiv v_1 w_1$, where $v_1 \in R$. In the opposite case if $\mathcal{A}$ passes the entire word w with an answer it does not belong to R, then the algorithm $\mathcal{B}$ answers w does not belong to R^* because every word from R^* is a concatenation of words from R. In the first case if w_1 is empty ($w_1 \equiv \Lambda$) the algorithm $\mathcal{B}$ answers $w \equiv v_1$ belongs to R and therefore it belongs to R^*. If w_1 is not empty, we apply the algorithm $\mathcal{A}$ to w_1 for which we will have two analogous cases. In the first one $w \equiv v_1 v_2 w_2$ where v_1 and v_2 belong to R ($v_1, v_2 \in R$). In the opposite one if $\mathcal{A}$ passes the entire word w_1 with an answer it does not belong to R, then the algorithm $\mathcal{B}$ answers w does not belong to R^* because every word from R^* is a concatenation of words from R.

This inductive process is finite because every next applying the algorithm $\mathcal{A}$ is to a shorter word. A very important note is this applying is in only one way which is determined by the property (1) from proposition 1. above: each word from R^* has only one presentation as a concatenation of words from R.

We will receive in this way finally a single presentation $w \equiv v_1 v_2 \ldots\ v_k w_k$ (where v_1, v_2, ..., v_k belong to R, i.e. $v_1, v_2, \ldots, v_k \in R$) for which a next applying the algorithm $\mathcal{A}$ can not separate a prefix from R in w_k. Therefore w_k is empty ($w_k \equiv \Lambda$) in the first case or w_k does not belong to R in the opposite second case when $\mathcal{A}$ passes it entirely with an answer it is not from R. In the first case the algorithm $\mathcal{B}$ gives an answer the word $w \equiv v_1 v_2 \ldots\ v_k$ belongs to R^*. In the second one its answer is the word $w \equiv v_1 v_2 \ldots\ v_k w_k$ does not belong to R^* because every word from R^* is a concatenation of words from R and there isn't a way to receive a such concatenation for w due to its only one presentation as a possible such concatenation according to the property (1). Therefore $\mathcal{B}$ is an effective procedure for recognizing belonging of any word from Σ^* to R^*, i.e. if R is recursive, then R^* is recursive too.

Conversely, let R^* is recursive. The proof R is recursive is the same as in [4] but based on the property (2) from proposition 1.: $MIN(R^*) = R$. We have an algorithm for recognizing membership of R^*. We test a given word and all its proper prefixes for this membership. According to the definition of the function MIN and the indicated property this word is from R if and only if when it belongs to R^* but no proper prefix of its belong to R^*. So R is recursive.

Lemma 2.3. *Let $\Gamma = (N, \Sigma, P, S)$ be a complete simple grammar which generates only words from the prefix-free language R in Σ^*. Then the generated by it language $L(\Gamma)$ covers the entire R, i.e. $L(\Gamma) = R$.*

Proof. Let w is an arbitrary word from R. We have to prove it can be generated by Γ. Due to the completeness of Γ (Definition 1.1) there are enough productions in P to continue a leftmost derivation of w. We will show no one derivation can end before or after the end of w. If a derivation ends before the last letter of w, then the derived proper prefix of its belongs to R which is in a contradiction with the given fact R is a prefix-free language. If a derivation ends after the last letter of w, then w from R would be a proper prefix of the derived word again from R which is again impossible. Therefore each derivation in Γ, which starts from the first letter of w, ends immediately after its last letter and therefore $w \in L(\Gamma)$.

A particular case of this statement is practically proved in [1], lemmas 5-8, but in a very long way and again for reduced word problem language of a group only. Probably this way has been a reason the proof to be repeated in the second part of the proof of lemma 5.1. [4] , but briefly inductively with respect to the length of the word and again for this particular language only. We will note we don't need here the property the generated by a simple grammar (simple) language is prefix-free in an arbitrary alphabet which is proved in [1], lemma 2. The above lemma 2.3. can be formulated in the following more expressive form:

Lemma 2.3'. *No one complete simple grammar can generate an absolute part of a prefix-free language.*

3. Entire Simple Grammars Generating Only Representatives of Any Fixed Element of a Group

In this section Σ is a double alphabet $\Sigma = \{x_1, x_2, \ldots, x_n; \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$. Let G be a finitely generated group with a monoid generating finite set Σ and with a finite set of defining relators. Let a be either one of the generators x_i or $\overline{x_i}$ ($a \equiv x_i$ or $a \equiv \overline{x_i}$, $i = 1, 2, \ldots, n$), but $\mathfrak{a}$ is the element of G whose a representative is

a (the symbol a is from Σ, the element $\mathfrak{a}$ is from G). We will construct many complete simple grammars with one and the same set Σ of terminal symbols, with different sets of non-terminal symbols, different initial symbols, and different sets of productions which will be assigned in one and the same way. The initial idea comes from [1], lemma 15, it was used in the same way in [4], lemma 5.1. Our modifications will be shown after the next proposition.

The basic part of the non-terminal symbols will be the set $\{..., n_{\mathfrak{a}}, n_{\mathfrak{a}^{-1}}, ...\}$ of $2n$ symbols which correspond to the $2n$ elements $\mathfrak{a}$ and $\mathfrak{a}^{-1}$ of the group with representatives the generators a and $\overline{a}$. We add to them a set $\{n_{\mathfrak{g}'}\}$ of symbols $n_{\mathfrak{g}'}$ which correspond to at one's own choosing chosen finite number other elements $\mathfrak{g}'$ of this group. The set $N_{\{\mathfrak{g}\}}$ of the non-terminal symbols will be their union, i.e.

$N_{\{\mathfrak{g}\}} = \{..., n_{\mathfrak{a}}, n_{\mathfrak{a}^{-1}}, ...\} \cup \{n_{\mathfrak{g}'}\} = \{n_{\mathfrak{g}}\}$.

We construct an obviously complete system P of simple productions in the following way: *for every* $n_{\mathfrak{g}} \in N_{\{\mathfrak{g}\}}$ *and every* $a \in \Sigma$

$n_{\mathfrak{g}} \to a$ *is a production if* $\mathfrak{g} = \mathfrak{a}$;

$n_{\mathfrak{g}} \to a\ n_{\mathfrak{a}^{-1}\mathfrak{g}}$ *is a production if* $\mathfrak{g} \neq \mathfrak{a}$ *and* $n_{\mathfrak{a}^{-1}\mathfrak{g}} \in N_{\{\mathfrak{g}\}}$;

If $n_{\mathfrak{1}}$ ($\mathfrak{g} = \mathfrak{1}$) is a terminal symbol, all productions of the form

$n_{\mathfrak{1}} \to a n_{\mathfrak{a}^{-1}}$

are among them because $\mathfrak{a} \neq \mathfrak{1}$

and $n_{\mathfrak{a}^{-1}.\mathfrak{1}} = n_{\mathfrak{a}^{-1}} \in N_{\{\mathfrak{g}\}}$.

$n_{\mathfrak{g}} \to a\ n_{\mathfrak{a}^{-1}} n_{\mathfrak{g}}$ *is a production if* $\mathfrak{g} \neq \mathfrak{a}$ *and* $n_{\mathfrak{a}^{-1}\mathfrak{g}} \notin N_{\{\mathfrak{g}\}}$.

The initial start symbol S of a such complete simple grammar $\Gamma = (N_{\{\mathfrak{g}\}}, \Sigma, P, S)$ can be any non-terminal symbol $n_{\mathfrak{g}_0}$, i.e. $S = n_{\mathfrak{g}_0}$ ($\mathfrak{g}_0$ is one of all participating elements $\mathfrak{g}$ of the group G).

Proposition 3.1. *All words derived from each non-terminal symbol* $n_{\mathfrak{g}}$ *of the just constructed complete simple grammar* $\Gamma = (N_{\{\mathfrak{g}\}}, \Sigma, P, S)$ *are representatives of the element* $\mathfrak{g}$ *in the group* G *with which* $n_{\mathfrak{g}}$ *is indexed.*

The proof is by induction with respect to the length of the derivation and repeats the first part of the proof of lemma 5.1 [4] and this of lemma 15 [1] but for more general grammars. Any derivation of length one in Γ is of the type $n_{\mathfrak{g}} \to a$ and by definition if $\mathfrak{g} = \mathfrak{a}$ only (a is a representative of $\mathfrak{g}$, i.e. $a \in \mathfrak{g}$). Let us assume the statement is true for all derivations of lengths no more than m, $m \geq 1$, starting from any non-terminal symbol. Any derivation from $n_{\mathfrak{g}}$ of length $m+1$ can start with one of the productions:

$n_{\mathfrak{g}} \to a\ n_{\mathfrak{a}^{-1}\mathfrak{g}}$ (*if* $\mathfrak{g} \neq \mathfrak{a}$ *and* $n_{\mathfrak{a}^{-1}\mathfrak{g}} \in N_{\{\mathfrak{g}\}}$) or

$n_{\mathfrak{g}} \to a n_{\mathfrak{a}^{-1}} n_{\mathfrak{g}}$ (*if* $\mathfrak{g} \neq \mathfrak{a}$ *and* $n_{\mathfrak{a}^{-1}\mathfrak{g}} \notin N_{\{\mathfrak{g}\}}$).

The lengths of the derivations which continue after those productions are no more than m and we can apply the inductive conjecture to them. So, in the first case the non-terminal $n_{\mathfrak{a}^{-1}\mathfrak{g}}$ on the right hand side derives a word u which belongs to $\mathfrak{a}^{-1}\mathfrak{g}$. The word au which will be derived from $n_{\mathfrak{g}}$ will belong to the element $\mathfrak{a}\mathfrak{a}^{-1}\mathfrak{g} = \mathfrak{g}$ of G, i.e. $au \in \mathfrak{g}$.

In the second case $n_{\mathfrak{a}^{-1}}$ on the right hand side derives a word s which is from $\mathfrak{a}^{-1}$ in G. The other non-terminal symbol $n_{\mathfrak{g}}$ there derives a word t which is from $\mathfrak{g}$. (The sum of the lengths of both last derivations is m.) The word ast which will be derived from $n_{\mathfrak{g}}$ will belong to $\mathfrak{a}\mathfrak{a}^{-1}\mathfrak{g}$ in G, i.e. $ast \in \mathfrak{a}\mathfrak{a}^{-1}\mathfrak{g} = \mathfrak{g}$.

This completes the inductive proof of the proposition.

$MIN(\mathfrak{g})$ is the set of all prefix free words from the element $\mathfrak{g}$ of the group G, i.e. the set of all words from $\mathfrak{g}$ which one of which has no proper prefix in it. The grammar from proposition 1. above is simple and complete. Therefore according to lemma 2.3. from section 2. this proposition has the following

Corollary 3.2. *For each element $\mathfrak{g}_0$ of the group G such that $n_{\mathfrak{g}_0}$ is a start symbol of the complete simple grammar $\Gamma = (N_{\{\mathfrak{g}\}}, \Sigma, P, S = n_{\mathfrak{g}_0})$ if the generated by this grammar representatives of $\mathfrak{g}_0$ contain in $MIN(\mathfrak{g}_0)$, then $L(\Gamma)$ coincides with the set $MIN(\mathfrak{g}_0)$ of all words from $\mathfrak{g}_0$ which have no proper prefix in it, i.e. $L(\Gamma) = MIN(\mathfrak{g}_0)$; briefly: if $L(\Gamma) \sqsubseteq MIN(\mathfrak{g}_0)$, then $L(\Gamma) = MIN(\mathfrak{g}_0)$.*

In particular, if $\mathfrak{g}_0 = \imath$ (then the start symbol is $S = n_\imath$), all words derivable by $\Gamma = (N_{\{\mathfrak{g}\}}, \Sigma, P, S = n_\imath)$ are from the word problem language of the group G. The simple grammars from lemma 15 [1] and lemma 5.1 [4] for theorem 5.2 [4] are constructed over a finite irreducible word problem language. They satisfy the conditions from corollary 3.2. here and we would have the proved there property from the indicated lemmas as a

Corollary 3.3. (**Theorem 5.2.** [4]) *If a finitely generated group has finite irreducible word problem language, then it has simple reduced word problem language.*

NOTE: We will show some of the modifications which we promised in the beginning of this section. First one is separating the elements $\mathfrak{a}$ and a^{-1} of the group whose representatives are the inverse generators a and $\overline{a}$. It is not necessary to prove especially a and $\overline{a}$ have inverse words, the corresponding products with which are from the irreducible word problem language of the group. That is correct simply because each one of them is obviously inverse to the other one and the words $a\overline{a}$ and $\overline{a}a$ are obviously irreducible. We can add arbitrary elements of the group to them and the same proof for generating only the indicated representatives goes. So, these modifications are significant because they lead to the just pointed freedom.

Statement 3.4. *Each group with a simple reduced word problem language with respect to some monoid generating set has a finite irreducible word problem language with respect to this generating set.*

This statement is an answer to question 5.3. with which the paper [4] ends. For its proof the notations of the type u^{-1} in lemmas 9. and 10. from [1] must simply be substituted by notations of the type $\overline{u}$ where the word $\overline{u}$ is the inverse word of u in a double alphabet.

4. Languages of Partial Order in a Group

Let G be a (monoid) finitely generated group with a generating set the double alphabet Σ as in the previous section. Let the partial order in G be presented by its positive cone $P = P^+ \cup \{\imath\}$. Here P^+ is the pure subsemigroup of the strongly positive elements, i.e. it is an invariant subsemigroup of the group which does not contain the unit element $\imath$. Let $\mathcal{P}$ and $\mathcal{P}^+$ be the full prototypes of P and P^+ correspondingly at the natural homomorphism φ of Σ^* over G, i. e. $\varphi^{-1}(P) = \mathcal{P}$ and $\varphi^{-1}(P^+) = \mathcal{P}^+$. Let $\mathcal{E} = \mathcal{P} \cap \overline{\mathcal{P}}$. Then $\varphi^{-1}(\imath) = \mathcal{E}$ (i.e. $\mathcal{E}$ is the word problem language of the group G). We can name $\mathcal{P}$ *a language of the positive cone in* G or *a positive cone language.*

The word problem language $\mathcal{E}$ in Σ^* satisfies two conditions from

Proposition 3.3. [4] *Let $\mathcal{E}$ be a subset of Σ^*; then $\mathcal{E}$ is the word problem of a group if and only if it satisfies the following conditions:*

(1) if $\alpha \in \Sigma^*$, then there exists $\beta \in \Sigma^*$ such that $\beta\alpha \in \mathcal{E}$ ($\alpha\beta \in \mathcal{E}$ stays here incorrectly in [4])*;*

(2) if $\alpha \in \mathcal{E}$ and $u\alpha v \in \mathcal{E}$, then $uv \in \mathcal{E}$.

We denote here the word problem language by $\mathcal{E}$ instead by W as it is in [4]. The reason for the marked correction is indicated in the section for the preliminary concepts.

Theorem 4.1. *Let G be a finitely generated group with a monoid generating set Σ and $\mathcal{E}$ be its word problem language which satisfies the indicated above conditions (1) and (2). If P^+ is the cone of the strongly positive elements of some partial order in G, then its full prototype $\mathcal{P}^+ = \varphi^{-1}(P^+)$ at the natural homomorphism φ satisfies the following conditions:*

(3) if $uv \in \mathcal{P} = \mathcal{P}^+ \cup \mathcal{E}$ and $\alpha \in \mathcal{P}$, then $u\alpha v \in \mathcal{P}$;

(4) if $\alpha \in \mathcal{P}^+$, then every β from (1), for which $\beta\alpha \in \mathcal{E}$, does not belong to $\mathcal{P}^+$.

Conversely, if some language $\mathcal{P}^+$ in Σ^* for which $\mathcal{P}^+ \cap \mathcal{E} = \varnothing$ satisfies the conditions (3) and (4), then it assigns a partial order in the group G with a word problem language $\mathcal{E}$.

Proof. Let G be a partially ordered group with generators Σ and with a cone P of the positive elements. Then the corresponding language of the positive cone in Σ^* is $\mathcal{P} = \varphi^{-1}(P)$. The language $\mathcal{E} = \mathcal{P} \cap \overline{\mathcal{P}}$ is exactly the word problem language of the group G and it satisfies the conditions (1) and (2). Let $uv \in \mathcal{P}$ and $\alpha \in \mathcal{P}$. Then $\varphi(uv) = \varphi(u)\varphi(v) = \mathfrak{u}\mathfrak{v} \in P$, i.e. $\mathfrak{u}\mathfrak{v} \geq \imath$ and $\varphi(\alpha) = \mathfrak{a} \in P$. From $\mathfrak{a} \geq \imath$ we receive $\mathfrak{u}\mathfrak{a}\mathfrak{v} \geq \mathfrak{u}\mathfrak{v}$ and therefore $\mathfrak{u}\mathfrak{a}\mathfrak{v} \geq \imath$. The last one means $uav \in \mathcal{P}$ because $\varphi(uav) = \mathfrak{u}\mathfrak{a}\mathfrak{v}$ with which the condition (3) is proved. The product of two strongly positive elements is again strongly positive from which it follows no one of the factors (in (4), where we suppose the opposite) can be inverse to the other one. This proves the last condition.

Conversely, suppose the language $\mathcal{P}^+ (\mathcal{P}^+ \cap \mathcal{E} = \varnothing)$ in Σ^* together with $\mathcal{E}$ satisfies the conditions (3) and (4). We construct a group G whose word problem language is $\mathcal{E}$ according to the indicated proposition from [4]. We will show the image $P^+ = \varphi(\mathcal{P} \setminus \mathcal{E})$ of $\mathcal{P}^+$ is a pure subsemigroup in G, i.e. P^+ assigns a partial order in it. We note before that the used in this proposition from [4] syntactic congruence

$\alpha_1 \ \sim \alpha_2 \Leftrightarrow (ua_1v \in \mathcal{E} \Longleftrightarrow u\alpha_2 v \in \mathcal{E}$ for arbitrary $u, v \in \Sigma^*)$ could be found in the cited in [5], [6] paper by Rabin M.D. and D. Scott and monograph by S. Ginsburg. It was shown in the section with preliminaries why a generalization of its was introduced and used by us. *in them.*

Two corollaries from the property (3) are valid: (3a) if $\alpha, \beta \in \mathcal{P}$, then $\alpha\beta \in \mathcal{P}$ and (3b) for every $u \in \Sigma^*$ and every $\alpha \in \mathcal{P}$ the conjugate word $\overline{u}\alpha u \in \mathcal{P}$. Really, (3a): For the empty word Λ and β the word $\Lambda\beta \in \mathcal{P}$. Then $\Lambda\alpha\beta \equiv \alpha\beta \in \mathcal{P}$; (3b) $\overline{u}u \in \mathcal{E}$ and $\alpha \in \mathcal{P}$. Then $\overline{u}u \in \mathcal{P}$ and $\overline{u}\alpha u \in \mathcal{P}$ after (3).

Our goal is to prove $P^+ = \varphi(\mathcal{P}^+)$ is a pure subsemigroup in the already constructed group G. Let $\mathfrak{a}, \mathfrak{b} \in P^+$. Then there exist prototypes $\alpha \in \mathcal{P}^+$, $\beta \in \mathcal{P}^+$ of theirs and $\alpha\beta \in \mathcal{P}$ due to (3a). It $(\alpha\beta)$ can't be from $\mathcal{E}$ due to the condition (4).

We will show $P^+ = \varphi(\mathcal{P}^+)$ is invariant. Let $\mathfrak{a}$ is an arbitrary element of P^+ and $\mathfrak{u}$ is an arbitrary element of the entire group G. Let $\alpha \in \mathcal{P}^+$ is a prototype of $\mathfrak{a}$ and $u \in \Sigma^*$ is a prototype of $\mathfrak{u}$. Then $\overline{u}u \in \mathcal{E}$ and from $\overline{u}u \in \mathcal{P}$ and $\alpha \in \mathcal{P}$ we receive $\overline{u}\alpha u \in \mathcal{P}$. Therefore $\mathfrak{u}^{-1}\mathfrak{a}\mathfrak{u} = \varphi(\overline{u}\alpha u) \in P$. It remains to prove $\mathfrak{u}^{-1}\mathfrak{a}\mathfrak{u} \neq \imath$. Really, if $\mathfrak{u}^{-1}\mathfrak{a}\mathfrak{u} = \imath$, then $\mathfrak{a} = \imath$ which contravenes $\mathfrak{a} \in P^+$.

The theorem is proved. We would like to pay attention to the condition (3). Obviously it is reversed to the condition (2) cited from [4] where it is proved (as a consequence of its) a word problem language satisfies (3) too. It would be interesting to prove or to disprove a

Conjecture 4.2. *The condition (2) is a consequence from the condition (3) in $\mathcal{E}$.*

section*References

[1] Haring-Smith, R.H. *Groups and Simple Languages*, Trans. Amer. Math. Soc. **1983**, 279. 337-356.

[2] L. Fucks, *Partially Ordered Algebraic Systems*, “Mir”, Moscow **1965** (Translation into Russian).

[3] M. Hall, Jr., *The Theory of groups*, The Macmillan Company, NY, **1959**.

[4] Parkes, D.W. and Thomas, R.M. *Groups with Context-Free Reduced Word Problem*, Communications in Algebra **2002**, 30(7). 3143-3156.

[5] Tarkalanov K., *The Context-free Languages and some Questions from Semigroup and Group Theory*, Annals Union of the Scientific Workers in Bulgaria, Br. Plovdiv, A Scientific Session of the young Scientific Workers, **1979**(**1980**). 131-142 (in Bulgarian).

[6] Tarkalanov K.D., *Connecting Algorithmic Problems in Semigroups with the Theories of Languages and Automata*, Studia Univ. “Babesh-Bolyai”, Mathematica, v. XLIV, No 1, March **1999**, 95-100.

Quincy College, 34 Coddington Street,Greater Boston, Quincy, MA 02169, USA
E-mail address: Ktarkalanov@hotmail.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

A COMPARISON OF CLUSTERING TECHNIQUES IN ASPECT MINING

GABRIELA ŞERBAN AND GRIGORETA SOFIA MOLDOVAN

Abstract. This paper aims at presenting and comparing three clustering algorithms in aspect mining: *k-means (KM)*, *fuzzy c-means (FCM)* and *hierarchical agglomerative clustering (HAC)*. Clustering is used in order to identify crosscutting concerns. We propose some quality measures in order to evaluate the results and we comparatively analyze the obtained results on two case studies.

Keywords: clustering, aspect mining.

1. Introduction

1.1. **Clustering.** Clustering is a division of data into groups of similar objects.

Clustering can be considered the most important *unsupervised learning* problem: so, as every other problem of this kind, it deals with finding a *structure* in a collection of unlabeled data.

Unsupervised classification, or clustering, aims to differentiate groups (classes or clusters) inside a given set of objects, with respect to a set of relevant characteristics or attributes of the analyzed objects. A *cluster* is, therefore, a collection of objects, which are similar between them and dissimilar to the objects belonging to other clusters.

Let $X = \{O_1, O_2, \ldots, O_n\}$ be the set of objects to be clustered. Using the vector-space model, each object is measured with respect to a set of l initial attributes $A_1, A_2, ..., A_l$ (a set of relevant characteristics of the analyzed objects) and is therefore described by a l-dimensional vector $O_i = (O_{i1}, \ldots, O_{il}), O_{ik} \in \Re, 1 \leq i \leq n, 1 \leq k \leq l$.

Received by the editors: May 24, 2006.

2000 *Mathematics Subject Classification.* 62H30, 68N99.

1998 *CR Categories and Descriptors.* I.5.3[**Computing Methodologies**]: Pattern Recognition – *Clustering*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement –*Restructuring, reverse engineering, and reengineering.*

The measure used for discriminating objects can be any *metric* or *semimetric* function (d). In our approach we have used the *Euclidian distance*:

$$d(O_i, O_j) = d_E(O_i, O_j) = \sqrt{\sum_{k=1}^{l}(O_{ik} - O_{jk})^2}$$

The *similarity* between two objects O_i and O_j is defined as

$$sim(O_i, O_j) = \frac{1}{d(O_i, O_j)}$$

Many clustering techniques are available in the literature. Most clustering algorithms are based on two popular techniques known as *partitional* and *hierarchical* clustering ([4], [6] and [7]).

1.2. **Aspect Mining.** The Aspect Oriented Programming (AOP) is a new paradigm that is used to design and implement *crosscutting concerns* [9]. A *crosscutting concern* is a feature of a software system that is spread all over the system, and whose implementation is tangled with other features' implementation. Logging, persistence, and connection pooling are well-known examples of crosscutting concerns. In order to design and implement a crosscutting concern, AOP introduces a new modularization unit called *aspect.* At compile time, the aspect is woven to generate the final system, using a special tool called *weaver.* Some of the benefits that the use of AOP brings to software engineering are: better modularization, higher productivity, software systems that are easier to maintain and evolve.

Aspect mining is a relatively new research direction that tries to identify crosscutting concerns in already developed software systems, without using AOP. The goal is to identify them and then to refactor them to aspects, in order to achieve a system that can be easily understood, maintained and modified.

Crosscutting concerns in non AO systems have two symptoms: *code scattering* and *code tangling. Code scattering* means that the code that implements a crosscutting concern is spread across the system, and *code tangling* means that the code that implements some concern is mixed with code from other (crosscutting) concerns.

1.3. **Related Work.** Many aspect mining techniques have been proposed so far ([2], [5], [12], [13], [15], [16], [17]). [5], [13] and [16] use clustering for identifying crosscutting concerns, but in different contexts.

In [13] we have proposed a clustering approach in aspect mining based on *k-means* and *hierarchical agglomerative* clustering. We have also defined in [14] a set of new quality measures in order to evaluate the results of clustering based aspect mining techniques. Based on the approach proposed in [13], this paper presents a comparison of three clustering algorithms: *k-means*, *fuzzy c-means* and *hierarchical agglomerative*, both from the aspect mining and clustering points of view.

The paper is structured as follows. Section 2 presents the context in which clustering is used in aspect mining. The clustering algorithms used in our comparison are described in Section 3. A comparative analysis of the results obtained on two case studies, based on some quality measures, is reported in Section 4. Section 5 presents some conclusions and further work.

2. Clustering approach in the context of aspect mining

In this section we present the problem of identifying *crosscutting concerns* as a clustering problem.

2.1. **Formal model.** Let $M = \{m_1, m_2, ..., m_n\}$ be the software system, where $m_i, 1 \leq i \leq n$ is a method of the system. We denote by n ($|M|$) the number of methods in the system.

We consider a crosscutting concern as a set of methods $C = \{c_1, c_2, ..., c_{cn}\}$, methods that implement this concern. The number of methods in the crosscutting concern C is $cn = |C|$. Let $CCC = \{C_1, C_2, ..., C_q\}$ be the set of all crosscutting concerns that exist in the system M. The number of crosscutting concerns in the system M is $q = |CCC|$.

Partition of a system M.
The set $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ is called a **partition** of the system M iff $1 \leq p \leq n$, $K_i \subseteq M, K_i \neq \emptyset, \forall i \in \{1, 2, ..., p\}$, $M = \bigcup\limits_{i=1}^{p} K_i$ and $K_i \cap K_j = \emptyset$, $\forall i, j \in \{1, 2, ..., p\}, i \neq j$.

In the following we will refer K_i as the i-th *cluster* of $\mathcal{K}$ and $\mathcal{K}$ as a *set of clusters*.

In fact, the problem of aspect mining can be viewed as the problem of finding a partition K of the system M.

2.2. **Identification of crosscutting concerns.** The steps for identifying the crosscutting concerns are as follows:

- **Computation** -Computation of the set of methods in the selected source code, and computation of the attributes set values, for each method in the set.
- **Filtering** - Methods belonging to some data structures classes like *ArrayList, Vector* are eliminated. We also eliminate the methods belonging to some built-in classes like *String, StringBuffer, StringBuilder, etc.*
- **Grouping** - The remaining set of methods is grouped into clusters using a clustering algorithm (*KM*, *FCM* or *HAC*, in this paper). The clusters

are sorted by the average distance from the point 0_l in descending order, where 0_l is the l dimensional vector with each component 0.
- **Analysis** - The clusters obtained are analyzed to discover which clusters contain methods belonging to crosscutting concerns. We analyze the clusters whose distance from 0_l point is greater than a threshold (eg. two).

3. Clustering Algorithms in Aspect Mining

In this section we briefly describe three clustering algorithms that we will use in the **grouping** step described in subsection 2.2, in order to identify a partition $\mathcal{K}$ of a system M.

In our approach, the objects to be clustered are the methods from the system $M = \{m_1, m_2, ...m_n\}$. The methods belong to the application classes or are called from the application classes.

We will consider each method as a l-dimensional vector: $m_i = (m_{i1}, \dots, m_{il})$.

In our approach we have considered two vector-space models:

- The vector associated with the method m is $\{FIV, CC\}$, where FIV is the fan-in value and CC is the number of calling classes. We denote this model by $\mathcal{M}_1$.
- The vector associated with the method m is $\{FIV, B_1, B_2, ...B_{l-1}\}$, where FIV is the fan-in value and B_i is the value of the attribute corresponding to the application class C_i. The value of B_i is 1, if the method M is called from a method belonging to C_i, and 0, otherwise. We denote this model by $\mathcal{M}_2$.

3.1. Hard k-means clustering (KM).

Hard k-means clustering is also known as *c-means* clustering. The *k-means* algorithm partitions the collection of n methods of the system M into k distinct and non-empty clusters. The partitioning process is iterative; it stops when a partition that minimizes the squared sum error (*SSE*) is achieved. The *SSE* of a partition $\mathcal{K}$ is defined as:

$$SSE(\mathcal{K}) = \sum_{j=1}^{p} \sum_{m_i^j \in K_j} d^2(m_i^j, f_j) \tag{1}$$

where the cluster K_j is a set of methods $\{m_1^j, m_2^j, ..., m_{n_j}^j\}$ and f_j is the centroid (mean) of K_j:

$$f_j = \left(\frac{\sum_{k=1}^{n_j} m_{k1}^j}{n_j}, \dots, \frac{\sum_{k=1}^{n_j} m_{kl}^j}{n_j} \right)$$

Hence, the *k-means* algorithm minimizes the intra-cluster distance. The algorithm starts with k initial centroids, then iteratively recalculates the clusters (each object is assigned to the closest cluster - centroid) and their centroids until convergence is achieved.

The main disadvantages of *k-means* are:

- The performance of the algorithm depends on the initial centroids. So the algorithm gives no guarantee for an optimal solution, corresponding to the global objective function minimum.
- The user needs to specify the number of clusters in advance.

In order to avoid these two main disadvantages of *k-means*, based on the approach presented in [13], we propose a new heuristic for choosing the number of clusters and the initial centroids. This heuristic will provide a good enough selection for the initial centroids.

We use the following *heuristic* for choosing the number of clusters and the initial centroids:

(i) The initial number k of clusters is n (the number of methods from the system).
(ii) The method chosen as the first centroid is the most "distant" method (the method that maximizes the sum of distances from the other methods).
(iii) The next centroid is chosen as the method that is the most distant from the nearest centroid already chosen, and this distance is strictly positive. If such a method does not exist, the number k of clusters will be decreased.
(iv) The step (iii) will be repeatedly performed, until k centroids will be reached.

3.2. **Fuzzy C-means Clustering (FCM).** *Fuzzy c-means clustering* ([1], [6]), also known as Fuzzy ISODATA, is a clustering technique which is separated from hard k-means that employs hard partitioning. *FCM* employs fuzzy partitioning such that a data point (method) can belong to all groups with different membership degrees between 0 and 1.

FCM is representative for the method of *overlapping* clustering. It uses fuzzy sets to cluster data, so each point may belong to two or more clusters with different degrees of membership. In this case, data will be associated to an appropriate membership value.

We will denote by k the number of clusters that we want to obtain in the data set, that was determined by applying *KM*. A membership matrix U is used, so that the equality below holds.

$$\sum_{i=1}^{k} U_{ij} = 1, \forall j \in \{1, 2, ..., n\}$$

In the above equation, U_{ij} $(i \in \{1,,2...,k\}, j \in \{1,,2...,n\})$ represents the membership degree of method j to cluster i.

By iteratively updating the cluster centers and the membership degrees for each method [1], *FCM* iteratively moves the cluster centers to the "right" location within the data set.

FCM does not ensure that it converges to an optimal solution, because the initial centroids (the initial values for matrix U) are randomly initialized.

FCM reports the final values for the matrix *U*. We propose the following equation:

$$K_i = \{j \mid j \in \{1, 2, ..., n\} \text{ and } U_{ij} > U_{rj}, \forall r \in \{1, 2, ..., n\}, r \neq j\},$$

in order to identify the clusters in data, after *FCM* was applied.

We mention that the number of clusters reported by *FCM* is less or equal to k (there is a possibility to obtain empty clusters).

3.3. Hierarchical Agglomerative Clustering (HAC).

The agglomerative (bottom-up) clustering methods begin with n singletons (sets with one element), merging them until a single cluster is obtained. At each step, the most similar two clusters are chosen for merging.

With the optimal number of clusters k determined after applying *KM*, we have applied a modified version of the traditional *HAC* algorithm in order to determine k clusters in data (the agglomerative algorithm stops when k clusters are reached).

4. Experimental Evaluation

In order to evaluate the results of the proposed clustering algorithms, we consider two case studies that are briefly described in Subsection 4.2. The obtained results are evaluated using three quality measures that are defined in Subsection 4.1.

4.1. Quality Measures.

In this section we propose quality measures for evaluating the results of clustering based aspect mining techniques. The first measure (*SSE*) evaluates a partition from the clustering point of view, and the last two measures (*PAM*, *ACC*) evaluate a partition from the aspect mining point of view. In the following, we denote by $|A|$ the cardinality of the set A.

Squared Sum Error of a partition - *SSE*. The *squared sum error* of a partition $\mathcal{K}$, denoted by *SSE*($\mathcal{K}$), is defined as in equation (1).

From the point of view of a clustering technique, smaller values for SSE indicate better partitions, meaning that SSE has to be minimized.

Percentage of analyzed methods for a partition - PAM [14].

Let us consider that the partition $\mathcal{K}$ is analyzed in the following order:
$K_1, K_2, ..., K_p$.

The percentage of analyzed methods for a partition $\mathcal{K}$ with respect to the set CCC, denoted by $PAM(CCC,\mathcal{K})$, is defined as:

$$PAM(CCC,\mathcal{K}) = \frac{\sum\limits_{i=1}^{q} pam(C_i,\mathcal{K})}{q}.$$

$pam(C,\mathcal{K})$ is the minimum percentage of the methods that need to be analyzed in the partition $\mathcal{K}$ to discover the crosscutting concern C and is defined as:

$$pam(C,\mathcal{K}) = \frac{\sum\limits_{j=1}^{i} |K_j|}{|M|}$$

where $i = min\{t \,|\, 1 \leq t \leq p \text{ and } K_t \cap C \neq \emptyset\}$ is the index of the first cluster in the partition $\mathcal{K}$ that contains methods from C.

$PAM(CCC,\mathcal{K})$ defines the percentage of the minimum number of methods that need to be analyzed in the partition in order to discover all crosscutting concern that are in the system M. We consider that a crosscutting concern was discovered the first time a method that implements it was analyzed.

Based on the definition, $PAM(CCC,\mathcal{K}) \in (0,1]$. If each $C \in CCC$ has one method in the first analyzed cluster K_1 of the partition $\mathcal{K}$, then $PAM(CCC,\mathcal{K}) = \frac{|K_1|}{|M|}$, otherwise $PAM(CCC,\mathcal{K}) > \frac{|K_1|}{|M|}$.

Smaller values for PAM indicate short time for analysis, meaning that PAM has to be minimized.

Accuracy of a clustering based aspect mining technique - ACC. Let $\mathcal{T}$ be a clustering based aspect mining technique.

The accuracy of $\mathcal{T}$ with respect to a partition $\mathcal{K}$ and the set CCC, denoted by $ACC(CCC,\mathcal{K},\mathcal{T})$, is defined as:

$$ACC(CCC,\mathcal{K},\mathcal{T}) = \frac{\sum\limits_{i=1}^{q} acc(C_i,\mathcal{K},\mathcal{T})}{q}.$$

$$acc(C,\mathcal{K},\mathcal{T}) = \begin{cases} \frac{|C \cap K_j|}{|C|} & \text{, if } K_j \text{ is the first cluster in which } C \text{ was discovered} \\ & \text{ by } \mathcal{T} \\ 0 & \text{, otherwise} \end{cases}$$

is the accuracy of $\mathcal{T}$ with respect to the crosscutting concern C. For a given crosscutting concern $C \in CCC$, $acc(C,\mathcal{K},\mathcal{T})$ defines the proportion of methods from C that appear in the first cluster where C was discovered.

In all clustering based aspect mining techniques, only a part of the clusters are analyzed, meaning that some crosscutting concerns may be missed.

Based on the above definition, $ACC(CCC, \mathcal{K}, \mathcal{T}) \in [0,1]$. $ACC(CCC, \mathcal{K}, \mathcal{T}) = 1$ iff $acc(C, \mathcal{K}, \mathcal{T}) = 1, \forall\ C \in CCC$. In all other situations, $ACC(CCC, \mathcal{K}, \mathcal{T}) < 1$.

Larger values for ACC indicate better partitions with respect to CCC, meaning that ACC has to be maximized.

4.2. **Case Studies.** In order to evaluate the results, we consider two case studies: Carla Laffra's implementation of Dijkstra algorithm [10] and JHotDraw, version 5.2 [8].

The first case study is a Java applet that implements Dijkstra algorithm in order to determine the shortest path in a graph. It was developed by Carla Laffra and consists in **6** classes and **153** methods.

The second case study is a Java GUI framework for technical and structured graphics, developed by Erich Gamma and Thomas Eggenschwiler, as a design exercise for using design patterns. It consists in **190** classes and **1963** methods.

4.3. **Comparative Analysis of the Results.** In this section we comparatively present the results obtained after applying the clustering methods described in Section 3 with respect to the quality measure described above, for the case studies presented in this section.

Case study	Clustering algorithm	Model	No. of clusters	PAM	ACC	SSE
Laffra	KM	$\mathcal{M}_1$	13	0.0964	0.6666	0
Laffra	KM	$\mathcal{M}_2$	22	0.1029	0.6666	0
Laffra	FCM	$\mathcal{M}_1$	11	0.0947	0.6666	2.9148
Laffra	FCM	$\mathcal{M}_2$	19	0.0996	0.6666	7.0921
Laffra	HAC	$\mathcal{M}_1$	13	0.1241	0.5000	18.8685
Laffra	HAC	$\mathcal{M}_2$	22	0.1307	0.5000	16.7567
JHotDraw	KM	$\mathcal{M}_1$	93	0.0736	0.2782	0
JHotDraw	KM	$\mathcal{M}_2$	586	0.0770	0.2782	0
JHotDraw	FCM	$\mathcal{M}_1$	89	0.0735	0.2782	10.9971
JHotDraw	FCM	$\mathcal{M}_2$	26	0.0342	0.7812	64084.92
JHotDraw	HAC	$\mathcal{M}_1$	93	0.0718	0.3095	267.3225
JHotDraw	HAC	$\mathcal{M}_2$	586	0.0811	0.2782	119.6836

TABLE 1. The values of the quality measures for the two case studies.

Table 1 presents the results obtained by applying the three clustering algorithms, for the two vector space models.

As presented in Subsection 4.1, the partitions that minimize the *squared sum error (SSE)* are better from the clustering point of view.

The conclusions reached after analyzing the obtained results from the aspect mining point of view, are presented below.

The analysis of the results based on the clustering algorithm used:

Laffra case study

- For all algorithms, vector space model $\mathcal{M}_1$ has provided better results.
- The best results were obtained using *FCM*.

JHotDraw case study

- Vector space model $\mathcal{M}_1$ has provided better results for *KM* and *HAC*, and vector space model $\mathcal{M}_2$ for *FCM*.
- The best results were obtained using *FCM*.

The analysis of the results based on the vector space model used:

Laffra case study

- For both vector space models, *FCM* has provided better results.

JHotDraw case study

- For vector space model $\mathcal{M}_1$, *HAC* has provided better results, and for vector space model $\mathcal{M}_2$, *FCM* has provided better results.

After analyzing the obtained results, we can conclude that the vector space models used in the proposed clustering based aspect mining technique should be improved. This can also be the cause of the lack of correlation between the results from the aspect mining point of view and from the clustering point of view.

5. Conclusions and Future Work

In this paper we have comparatively presented the results of applying three clustering algorithms in aspect mining. The comparison was made mostly from the aspect mining point of view, using a set of quality measures (*SSE*, *PAM*, *ACC*).

Further work can be done in the following directions:

- To improve the vector-space models used in this clustering based aspect mining approach. In our opinion, the vector space models have significantly influenced the obtained partitions.
- To compare, from the aspect mining point of view, the results obtained by the clustering algorithms proposed in this paper with other clustering approaches that were proposed in the literature (such as variable selection for hierarchical clustering [3], search based clustering [11]).
- To apply this approach for other case studies like PetStore and JEdit.

References

[1] S. Albayrak and F. Amasyali. Fuzzy c-means clustering on medical diagnostic systems. In *Turkish Symposium on Artificial Intelligence and Neural Networks - TAINN*, 2003.

[2] S. Breu and J. Krinke. Aspect Mining Using Event Traces. In *Proc. International Conference on Automated Software Engineering (ASE)*, pages 310–315, 2004.

[3] E. B. Fowlkes, G. Gnanadesikan, and J. R. Kettering. *Design, Data, and Analysis: By Some Friends of Cuthbert Daniel.* Wiley, New York, NY, 1987.

[4] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2001.

[5] L. He and H. Bai. Aspect Mining using Clustering and Association Rule Method. *International Journal of Computer Science and Network Security*, 6(2A):247–251, February 2006.

[6] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[7] A. Jain, M. N. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[8] JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[9] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[10] C. Laffra. Dijkstra's Shortest Path Algorithm. http://carbon.cudenver.edu/ hgreenbe/ courses/dijkstra/DijkstraApplet.html.

[11] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, pages 50–59. IEEE Computer Society, 1999.

[12] M. Marin, A. van, Deursen, and L. Moonen. Identifying Aspects Using Fan-in Analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004)*, pages 132–141. IEEE Computer Society, 2004.

[13] G. S. Moldovan and G. Serban. Aspect Mining using a Vector-Space Model Based Clustering Approach. In *Proceedings of Linking Aspect Technology and Evolution Workshop(LATE 2006)*, Bonn, Germany, March 2006.

[14] G. S. Moldovan and G. Serban. Quality Measures for Evaluating the Results of Clustering Based Aspect Mining Techniques. In *Proceedings of Towards Evaluation of Aspect Mining(TEAM), ECOOP*, 2006, to be published.

[15] Orlando Alejo Mendez Morales. Aspect Mining Using Clone Detection. Master's thesis, Delft University of Technology, The Netherlands, August 2004.

[16] D. Shepherd and L. Pollock. Interfaces, Aspects, and Views. In *Proceedings of Linking Aspect Technology and Evolution Workshop(LATE 2005)*, March 2005.

[17] P. Tonella and M. Ceccato. Aspect Mining through the Formal Concept Analysis of Execution Traces. In *Proceedings of the IEEE Eleventh Working Conference on Reverse Engineering (WCRE 2004)*, pages 112–121, November 2004.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: grigo@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

COMPONENTS MODELING IN UML 2

JEAN-MICHEL BRUEL AND ILEANA OBER

Abstract. The Object Management Group (OMG) has recently recommended for adoption the proposal that will be the basis for the new version of the Unified Modeling Language – UML. In this version 2.0 of UML, the component model has been completely modified. The first goal of this paper is to describe this new model. The second goal is to study if it is really more than a notation improvement. We go through the requirements for a general component modeling language, the existing efforts in this area, and the new UML 2.0 model itself.

1. Introduction

Nowadays, even the simplest software systems are complex. Generalization of the client/server architecture, importance of the notion of services, distributed in different nodes of the system, and distantly available, real-time and critical aspects more and more involved, are some of the reasons for such a complexity. As a matter of fact there is a big issue in the use of reusable pieces of software components. But when it comes with components and composition, there are two complementary problems to solve [12]: "[...] how to meet requirements using component-based designs and how to design components that work well together."

Components and composition have been an active research topic for a long time now. But it is undoubtable that there is no such a usage of software components as in hardware for example. One of the reason advocated by international surveys is the lack of support for composition description and evaluation [24]. Not so long ago, when you typed "Component Modeling Language" in a web-search engine, not only you did not find so many links, but you found among the first ones a link to the Rational Rose web page. Looking through this page, and also through the recently adopted U2-partners proposal for the upcoming 2.0 version of the UML, it seems that UML 2.0 aims to be, among other things, a component modeling language [23].

Received by the editors: June 10, 2006.

2000 *Mathematics Subject Classification.* 68N99.

1998 *CR Categories and Descriptors.* D2.2 [**Software Engineering**]: Design Tools and Techniques – *Object-oriented design methods*.

We do not aim here to fully explore the requirements for a potential component modeling language (see e.g., [19, 2] for such requirements), but we intend to give a critical overview of the new UML support for components using the following organization: in section 2 we will describe composition models and languages; in section 3 we will describe the UML 2.0 component model; in section 4, we will present ongoing efforts that aim at providing good support for composition; in section 5 we discuss this model from a critical point of view; and we will conclude in section 6.

2. Composition models and languages

Component based development offers a means to deal with the increasing software complexity. We do not start with the rather philosophical and surely controversial question on what it is a component. Instead we will focus on the main issues related to components.

According to various authors [18, 25] components are characterized by their *interfaces*, which specify their possible interaction with their environment and offer a basis for establishing the compatibility between two or several components. The *provided interfaces* specify how a component can be used and summarize what it offers to the outside world. Besides the provided interfaces, a component should specify what the deployed environment should provide in order for the component to be able to offer the functionality specified in the provided interface [20]. These context dependencies are specified using *required interfaces*.

For a good component specification, the structural specification of components done in terms of provided and required interfaces should be complemented with a *component contract* [6, 17]. Specifying required interfaces is a first step in this direction, however it only covers the structural/signature level. More advanced techniques for component specification include invariants, pre- and post-conditions of the services offered/required by the interfaces, various kinds of logics, automata-like specification techniques for component abstractions, etc. The nature of the component contract, and of the contract specification technique varies depending on whether the component is viewed as a unit with state (as argued in [18]), or a stateless unit (as argued in [25]).

The component contract can be specified at various levels of abstraction and can address various needs.

[6] defines a four level contract framework, where each level corresponds to a class of contracts (from the first to the forth): syntactic, behavioral – described in terms of pre- and post-conditions and invariants, synchronization, and quality of service. In this framework, contracts corresponding to the first two levels can be checked statically (provided that enough verification theoretical background exists), while the last two levels are basically focused on dynamic behavior and on the integration of the component on the execution platform.

One important and much studied issue in this context is the component *composition*: gathering together various components into a working system. Related to this, lots of efforts address the impact of composition on the *properties* of the composed components. A thorough analysis of the composition, formalizing it as an operation that considers the components and their integration constraints can be found in [14]. The paper also proposes a framework for component composition.

In what follows we present a list of expected features needed for a good components model. This list is established independently of UML, and will serve as as a basis for discussing the properties of the component model offered by UML.

Precise modeling concepts. Since the component modeling offers an abstract view of the system under modeling, it is crucial to have unambiguous definitions and a precise semantics of the basic modeling concepts.

Expressive power. For this the parts that need to be covered [15] are: (i) interface specification: it should be possible to associate sets of interfaces to a component specification and to specify their relationship with the component, i.e. whether they are required or offered; (ii) semantic specification: define and document semantic information about the component's operations; (iii) extra-functional properties specification: such as quality of service (QoS) and timing.

Modeling features. [9] defines several key requirements for a component model. Among the most general we mention: (i) encapsulation and identity; (ii) composition: the model should not only support dynamic composition, but also support different semantics of composition; (iii) life-cycle: a general model should support different forms of components throughout the entire development life-cycle.

3. UML 2.0 COMPONENT MODEL

Backwards compatibility and the possibility to include architecture representation into UML models have been the key concerns in the definition of the new UML version. The component model itself has been improved, the overall concept of composition has been integrated, even at class level. We will first introduce the concepts and then discuss their pluses and minuses.

3.1. New concepts and improvements.

Justifying the drawbacks in the UML 1.5 version in terms of composition is out of the scope of this paper. For an introductory illustration of some major difficulties in using the UML 1.5 composition model, see [7]. UML 2.0 [23] provides support for decomposition through the new notion of structured classifiers[1]. A structured classifier is a classifier that can be internally decomposed (Classes, Collaboration, and Components). New constructs to support decomposition have been introduced: Part, Connectors, and Ports. Note that Part is a new name, but not a new concept. UML 2.0 allows the specification of

[1]In this paper, we use Sans Serif font to highlight the new UML 2.0 concepts.

physical components such as in UML 1.5, of logical components, i.e. specification level components (e.g., business components, process components) as well as deployed components (such as artifacts and nodes). A component is viewed as a "*self contained unit that encapsulates state + behavior of a set of classifiers*". It may have its own behavior specification and specifies a contract of provided/required services, through the definition of ports. It is hence a substitutable unit that can be replaced as long as port definitions do not change. Notice that the notion of "change" here is not defined and has to be taken at a syntactic level only. Notice also that ports are not reserved to components, but are available for any structured classifier.

Three new constructs are part of the component model. Note that those constructs can be used together with any composite diagram. These new concepts are:

Part: something that is internal to a composite structure. This is not much different from UML 1.5 except that aggregation applies to properties as well as association ends. This is due to the unification of attributes and associations. Notice that instances (of a class) and parts have similar notations, which might be confusing. The part names are not objects identifier, but role names. Parts have to be seen as roles, and instances as the realization that satisfy these roles.

Connector: expresses the relationship between parts and between ports. It is a link (an instance of association) that enables communication between two or more instances, in addition to everything that ordinary links enable (e.g., navigating to a neighboring object to retrieve a property from it, modify it, destroy it, etc.). It may be realized by pointer, network connection, etc.

Port: a kind of part, but mainly used to represent the connection point via which messages are sent to/received by a component (or a class). Ports have a type which is given by a set of interfaces (provided and required) and can be described with a state machine. UML 2.0 has introduced a specific kind of state machines (that describes usage protocols of parts of the system): protocol state machines, and hence renamed the previous state machine (for describing the behavior of some entities): behavioral state machine.

The interface represents a signature given in terms of a set of public features (operations, attributes, signals). Interface attributes as well as association between interfaces is new in UML 2.0. The interface use has been extended from UML 1.5: a classifier or a port may *implement or require* an interface, in addition to providing an interface. Interfaces can be attached to ports. A required interface attached to a port characterizes the behavioral features that the owning classifier expects from its environment via the given port, while a provided interface attached to a port characterizes the behavioral features offered by the owning classifier via the given port. Note the distinction between a port and an interface: an interface specifies a service offered/required by a classifier, while a port specifies the services offered/ required by the classifier via that particular interaction point (port). It is possible to attach to a port or to an interface, a protocol state machine that allows the definition of a more precise external view by making dynamic constraints on the sequence of operation calls and signal exchanges explicit. The protocol state machine of a port (if present) shall be compatible with the protocol state machines of all interfaces attached to it. However, this "compatibility" notion is not defined in the proposal.

3.2. **Component Diagrams.** In UML 1.5, component diagrams were support for physical components only. In UML 2.0, extends component diagrams from addressing physical components to logical ones. Also it allows component-based software engineering CBSE as it is now possible to trace from logical to physical components.

There is two possible views for components models: (i) an external one ("black box" view), where the focus is on contracts linking the component to its clients in terms of provided services; (ii) an internal view ("white box" view), hidden from the clients, where the focus is on how the component is organized in terms of parts, sub-components, connectors, etc.

There is two specific connectors for components: (i) an assembly connector is the link between a required (socket) and a provided (lollipop) interface of the same time; (ii) a delegation connector connects a port on the container to/from an internal port/part that has a compatible interface of the same kind (both provided or both required). An arrow indicate the delegation direction. To be more precise, connectors are between parts/ports that have compatible interfaces of different kinds (one provided, one required). It is one way to wire components together (the other way is to use dependencies as in UML 1.5 version, but these do not have an instance level counterpart).

3.3. **Support for composition.** New constructs and new approaches have been introduced in UML 2.0 with direct impact on the support for composition. To describe the links between a composite and its sub-components, UML 2.0 uses several notions of components. BasicComponents and PackagingComponents are capabilities of components modeled in separate packages for convenience of implementation. PackagingComponents are an extension of BasicComponents to define the grouping aspects of packaging components. A basic component inside a packaging component is informally called a *nested component*. Note that this notion is different from the one of a part, which is not specific to components, and which define an element of an internal structure. As we have shown, the internal structure of a component can be described by a component diagram. In fact, despite the added/modified notations for components constructs, the main change, or the one to most impact the ongoing research (such as ours [3]) is the introduction of the StructuredClass first class element, and, at a lower level, the new distinction of required interface. Components communicate together via messages going through their ports, using the same idea as processes in SDL [11] or capsules in ROOM [22]. Note that components can also communicate directly point to point, using the same schema as in the Corba Component Model (CCM [10]). Nevertheless it has to be conclude that components add only a little to structured classes as far as composition is concerned.

4. Existing efforts

Several working directions projects address the issue of UML components. We can classify them into the following categories:

- commercial tools - the various existing commercial UML tools offer support for compositional modeling in UML to various extents, most often on the model editing level. Few tools, such as Rhapsody,Telelogic Tau 2, etc. offer support for more advanced component related treatment such as component specific code generation, verifications, symbolic execution, etc.
- standardization - related efforts: EDOC and SPT.
- research projects ACCORD, AIT-WOODDES, OMEGA, CML etc.

In what follows we give some more details on the efforts mentioned at the previous last two points.

The **EDOC** (Enterprise Distributed Object Computing) [21] has a standard profile for UML, which introduces the notion of Component Collaboration Architecture (CCA), where some concepts such as process components, ports and connections are defined.

The UML Profile for Schedulability, Performance and Time (**SPT**) [22] offers no particular treatment to components, although it is implied that the component approach is suitable for the real-time domain. When describing concurrency, components are mentioned as one of the different kinds of *concurrent units*.

ACCORD is a French project aiming to define a Platform Independent component model, based on UML [27]. The proposed model uses the extension mechanisms of UML 1.5 to support concepts such as components, ports and connectors. In this model, component operations are attached to ports and the architectural constraints are expressed in the means of OCL meta-level constraints. One of the main contributions of this project is that to illustrate the derivation of a general model towards Platform Specific Models (PSMs) such as CCM or EJB.

Related to ACCORD, the **AIT-WOODDES** project focused on the specification of embedded real-time systems. One of the results of the project is a specific component model [26] using concepts developed in ACCORD project with specific real-time aspects.

The IST project **OMEGA** aimed to develop a methodology for modeling real-time and embedded systems in UML, with the aid of formal techniques. In this context, the efforts go towards both finding the best theoretical framework for verification in this precise context, and towards implementing them into tool sets developed on the top of UML commercial tools. In this context, a particular attention is given to the use of an appropriate component model, important for enforcing property-preserving refinement and for enabling efficient validation. The OMEGA component model is based inspired from the UML 2.0 component model, however the tool sets was built upon UML 1.5 tools, for commercial tool availability reasons.

The **CML** project [8] is an ongoing project aiming at the definition of a general purpose component model, based on the UML notation, and on a formal framework based on the Whole-Part Relationship (WPR). Two notions of composability are used: *horizontal composability* - component binding and cooperation in distributed systems and *vertical composability* - whole (container) components process requests of client components. Part components are fully encapsulated (into whole components) and are not units of deployment in this particular context, they provide implementations for the components that they belong to by means of delegation. This approach [4] proposes to extend the semantic properties of the whole-part relationship [3] by adapting its formal base to software composition.

5. UML 2.0 COMPONENTS ANALYZED FROM DIFFERENT PERSPECTIVES

5.1. **Critical point of view on the new UML 2.0 artifacts.** UML 2.0 provides useful notation artifacts that were missing in UML 1.5 [7]. We believe those improvements are not enough. This is why we have developed our own approach of composition using UML 2.0 [4], and why certainly others will do the same in the near future. Our theoretical framework is based on an adaptation to CBSE of a formalization of the whole-part relationship (WPR) [3]. We have extended the semantic properties of the WPR by adapting its formal base to software composition. We have determined which properties apply to component composition and defined formally these properties. We ground our approach on metamodeling and assertions in order to constrain the specification of composition relationships. Constraints are added to components at implementation time via generated contracts. In our approach, the notion of Whole component can be linked to the one of PackagingComponent and the Part components to the BasicComponent one. The benefits of our approach is the well definition of precise properties to characterize composition. For details and illustration of our approach, see [4, 5].

5.2. **Support for composition.** The requirements of the OMG RFP were explicit in terms of CBSE support: support for component assembly and plug-substitutability, support for the specification of common interaction patterns that might occur between components, support for modeling of component execution contexts (e.g., EJB and CCM containers), and support for profiles definitions for specific component architectures models (such as EJB, CORBA, and COM+). The effort of the proposal to address these requirements is undoubtable as we have described in section 3. Most of the construct needed for CBSE support have been introduced. A set of recommendation had been produced based on the several UML 2.0 RFI responses. In terms of composition, they were mainly: (i) improve the semantics and notation to support component-based development, (ii) better support for interfaces, and (iii) usage protocol for interfaces formalization. As we have mentioned before, the first point have been missed, in our opinion, in the sense that there is no semantic provided for the added constructs.

To summarize the pluses of the new UML 2.0 constructs we could say that it provides: means to express architecture, means to express component contracts, and more expressive communication description. And in terms of the minuses, we could say that: there is too many overlapping constructs, relationship between the various constructs are not clear, and the semantics has too many traps (e.g., misuse of new concepts) and lacks (e.g., of explanation and illustration of the usage of new concepts, informal assumptions, ...).

5.3. **Support for Agile Modeling.** As an illustration of the concrete use of the improvements brought by the new version, let us mention some of those made by Scott Ambler in its overall *Agile Modeling* approach [1]. Here are some examples of his proposal. In the Component Diagram, Ambler suggests that the ports should be linked to one interface only. This seems to be only informally assumed in some of the UML 2 spec, but there is no explicit restriction of the number of interfaces on a port. This simplifies the delegation mechanisms (internal structure). Three different relationships between ports and parts have been identified: (i) delegates, which is the usual relationship between a port and a part in UML 2.0. (ii) stereotyped delegates, which specific notation comes in replacement of the previous in order to prevent confusion with the unidirectional association, which has the very same drawing. (iii) realizes, which indicates that it is the realization of a port. Ports are viewed in this case as logical modeling constructs realized by physical constructs such as classes.

Another example is the use of class to systematically implements ports (using the *Façade* design pattern [13]). These classes implement the public operations required by the interfaces.

The main conclusion of the use of UML 2.0 from an *agile* point of view is the fact that it eases the application of the heuristics that were already defined in terms of good practices by supplying new and useful constructs (differentiation between application, infrastructure and domain components, definition of component contracts, etc.).

5.4. **Expressive power of the architecture description.** [16] presents a deep analysis of UML 1.5's expressive power for modeling software architecture, in a way natural to the way this is done in traditional architecture description languages. In the followings we start from the results of this analysis, and try to see how they are affected by the changes made in UML 2.0, and to what extent UML 2.0 answers or not the specific needs of software architecture modeling.

In [16], Medvidovic *et al.* identify a minimum set of requirements for evaluating the ability of UML to be used in software architecture modeling efficiently. In order to do this the authors apply two approaches for supporting architectural concerns within UML: the first is based on using UML "as it is", and the other on using standard UML extension mechanisms. The results of this study are that UML can be used to address architectural concerns, although this requires some extra-efforts and its it has some drawbacks when compared to the use of classical ADL for the same purpose.

The general conclusion is that the extensible design of UML renders it applicable to system architecture modeling. However, as the language was not primarily designed for this, there are some drawbacks in using UML for this purpose. Some architecture modeling specific concepts are missing, and they need to be added in UML, e.g. using extension mechanisms. As a result, the rules of architectural style (present in ADLs and maintained by ADL supporting tools) have to be managed and applied by the modeler and need to be documented in addition to the system/architecture modeling. This makes the design more difficult to manage and to maintain.

Another conclusion regards the modeling of behavior and interactions in the architecture modelled using UML 1.5. This is possible using sequence diagrams, collaboration diagrams, or state machine diagrams. However it is hard to establish the relationship between the specified behavior and the architectural elements, and it is hard to ensure that the intended behavior is correctly modelled in UML.

Apart of the general points mentioned before, we extract here those that seemed to us among the most noteworthy with respect to UML 1.5 suitability for architecture modeling:

(1) For UML 1.5 classes it is only possible to specify the list of events it can receive, not of those that can be sent. This is different from the common practice when using ADLs;
(2) As UML was primarily designed to address various kinds of concerns, although it can handle architecture specific needs, architecture modelers find the support for these aspects partially sufficient;
(3) Some of the software architecture-specific concepts are conceptually different of those existing in UML (and more generally in object-oriented design). It is the case for example of connectors. They can indeed be abstracted by a UML class (e.g., by using stereotypes), however some of the properties ADL connector's properties need to be explicitly modelled in UML. This is the case of some ADL, where connector interfaces are

context reflective. In UML this needs to be explicitly modelled, which makes the use of connectors in UML 1.5 heavier.

(4) The UML tool support do offer (as far as the authors of the cited paper and ourselves are aware) some of the features regularly offered by ADL supporting tools. In particular, the part concerning the infrastructure of the systems modelled (that enforces the desired topology, interface and interactions between system components) is not covered by general UML tools, and not offered by other UML supporting tools.

In the study above mentioned, among the weaknesses of UML 1.5 in supporting architecture modeling was the lack of some software architecture specific concepts. The situation changed a bit with the new UML 2.0, as some of these concepts are added to the language definition. In the followings, we will discuss on how much this impacted on the language ability to be used in architectural description.

The add of concepts like *port, port instance, connector*, and *architecture diagram* enriches the expressive power of UML, when it comes to architecture modeling. Indeed, having these concepts explicitly at language level (not having to model each of them) increases the readability of architecture designs done with UML, and avoids the use of conventions. However, as for the moment only very limited tool support exists for UML 2.0, it is hard to assess the impact of this improvement on the architectural modeling: how well the UML tools will manage the desired topologies, interfaces and interactions between system components.

A step forward was also done towards modeling the behavior and interactions in architecture modeling, by adding the possibility to specify/constrain the behavior as observed at ports or interfaces through protocol state machines. We see however a problem in the abundance of means existing in UML 2.0 to specify behavior and interactions in architecture, especially as the relationship between the various means is not clearly specified in the standard. Without a good modeling methodology and the right tool support, we fear that the new additions will not be used, due to the risk of confusion.

If we look at the list of more precise points raised on the UML 1.5 ability to address architecture modeling needs, we notice that some of them (point 1 and 3) are partially solved by adding new concepts to UML. Although it is still not possible to specify what events can an object generate to its environment, using implemented and required interfaces, it is possible to give information on both directions of the interaction of an object with its context.

Our discussion represents a first level analysis on the use of UML 2.0 for software architecture modeling, having as starting point the extensive study performed on UML 1.5 in [16], and the new UML 2.0 standard. This allowed us to infer the effect of the UML evolution. A deeper analysis, similar to the one performed for UML 1.5 is probably needed to correctly asses all the details of the UML 2.0 ability to address architecture modeling.

6. Conclusion

This paper aimed at discussing whether the recently adopted UML 2.0 proposal was good or not in terms of composition support. We have presented in this paper a brief overview of the new UML component model and we have done our best to found some good arguments in favor of the affirmative answer to this question. It is undoubtable that the improvements at the notation level will be useful for component-based systems developers. The new component model is more expressive and flexible than the one offered in the old version of the language. The main requests formulated in the RFI have been answered. However, some points still remain to be clarified, especially when it comes to the combined use of various concepts. This may require a non minor amount of work, if one wants to keep using all the concepts offered by UML 2.0. The application of the UML 2.0 on concrete case studies will tell how well new UML component model is adapted to the industry.

A step forward on the way of applying UML on system engineering is represented by the SysML initiative. SysML is a modeling language for systems engineering applications called Systems Modeling Language. SysML will customize UML 2.0 in order to support the specification, analysis, design, verification and validation of complex systems that include hardware and software components. It will be interesting to also follow what the people who already had some proposals for UML support for components, as presented in section 4, will answer to this same question. This might lead to interesting developments in this area.

References

[1] Scott W. Ambler. The Official Agile Modeling Site – The Diagrams of UML 2. Available at `http://www.agilemodeling.com`, 2003.

[2] ARTIST. Component-based Design and Integration Platforms : Roadmap. Technical Report W1.A2.N1.Y1, Project IST-2001-34820, 2003.

[3] Franck Barbier, Brian Henderson-Sellers, Annig Le Parc-Lacayrelle, and Jean-Michel Bruel. Formalization of the Whole-Part Relationship in the Unified Modeling Language. *IEEE Transactions on Software Engineering*, 29(5):459–470, May 2003.

[4] Nicolas Belloir, Jean-Michel Bruel, and Franck Barbier. Whole-Part Relationships for Software Component Combination. In Gerhard Chroust and Christian Hofer, editors, *Proceedings of the 29th Euromicro Conference on Component-Based Software Engineering*, pages 86–91. IEEE Computer Society Press, September 2003.

[5] Nicolas Belloir, Fabien Roméo, and Jean-Michel Bruel. Whole-Part based Composition Approach: a Case Study. In *Proceedings of the 30th Euromicro Conference – Component-Based Software Engineering Track (Euromicro'2004)*, March 2004. To be published.

[6] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. *IEEE Computer*, 13(7):38–45, 1999.

[7] Conrad Bock. UML 2 Composition Model. *Journal of Object Technology*, 3(10):47–73, November-December 2004.

[8] Jean-Michel Bruel. CML – Component Modeling Language: project proposal. Technical report, French National Sciences Funds, 2003.

[9] E. Bruneton, T. Coupaye, and Jean-Bernard Stefani. Recursive and Dynamic Software Composition with Sharing. In *Proceedings of 7th International Workshop on Component-Oriented Programming – WCOP02 at ECOOP 2002*, June 2002.

[10] CCM. CORBA Component Model. OMG Report ptc/02-08-03. URL: `http://www.omg.org/`.

[11] Laurent Doldi. *UML 2 Illustrated – Developing Real-Time & Communications Systems.* TMSO, October 2003.

[12] Desmond D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks With UML: The Catalysis Approach.* Addison-Wesley, 1998.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[14] Gregor Goessler and Joseph Sifakis. Composition for Component-Based Modeling. In *Proceedings of FMCO, November 5-8, 2002*, volume 2852. LNCS, 2003.

[15] Frank Lüders, Kung-Kiu Lau, and Shui-Ming Ho. *Building reliable component-based software systems*, chapter Specification of Software Components, pages 23–38. Number 2. Artech House Publishers, Boston, 2002.

[16] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, 11(1), 2002.

[17] Bertrand Meyer. Contracts for components. *Software Development Online*, July, 2000.

[18] Bertrand Meyer. The grand challenge of trusted components. In *ICSE 25, Portland, Oregon, May 2003.* IEEE Computer Press, 2003.

[19] Oscar Nierstrasz and Theo Dirk Meijler. Requirements for a Composition Language. In *Proceedings of ECOOP 94 workshop on Models and Languages for Coordination of Parallelism ad Distribution*, LNCS, pages 147–161. Springer Verlag, 1994.

[20] A. Olafsson and D. Bryan. On the need for required interfaces to components. In *Special Issues in Object-Oriented Programming – ECOOP 96 Workshop Reader*, pages 159–171. dpunkt Verlag, Heidelberg, 1997.

[21] OMG. UML Profile for Enterprise Distributed Object Computing Specification (EDOC). OMG document, Object Management Group, may 2002.

[22] OMG. UML Profile for Schedulability, Performance, and Time Specification (PST), Draft Adopted Specification. OMG document, Object Management Group, January 2002.

[23] OMG. Unified Modeling Language: Infrastructure and Superstructure, version 2.0. OMG document formal/05-07-05 and formal/05-07-04, Object Management Group, March 2006.

[24] High Confidence Software and Systems Coordinating Group. High Confidence Software and Systems Research Needs. Technical report, Interagency Working Group on Information Technology Research and Development, january 2001.

[25] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, ACM Press, NY, 1999.

[26] Francois Terrier and Sébastien Gérard. UML 2: Component model and RT feedback of AIT-WOODDES project to the U2 proposal. In *Workshop SIVOES-MONA - UML'2002, Dresden*, 2002.

[27] Tewfik Ziadi, Bruno Traverson, and Jean-Marc Jézéquel. From a UML Platform Independent Component Model to Platform Specific Component Models. In Jean Bezivin and Robert France, editors, *Workshop in Software Model Engineering*, 2002.

LIUPPA, Université de Pau et des Pays de l'Adour, 64000 Pau, France
E-mail address: `Jean-Michel.Bruel@univ-pau.fr`

IRIT, Université Paul Sabatier, Toulouse, France
E-mail address: `ileana.ober@irit.fr`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LI, Number 1, 2006

A UNIFORM ANALYSIS OF LISTS BASED ON A GENERAL NON-RECURSIVE DEFINITION

VIRGINIA NICULESCU

Abstract. The paper presents a general, non-recursive definition of lists, in order to be used as a starting point for a uniform analysis of them. A general parameterized abstract data type *List* is defined, based on the type parameter *Position*, and type parameter *TE* (the type of elements contained in the list). By instantiating the parameter *Position* to the concrete types *Index*, *SNode* and *DNode* we obtained the abstract data types: *IndexedList*, *SinglyLinkedList*, and *DoublyLinkedList*. For them different representations could be considered.

This definition that starts from a general parameterized ADT has the advantage of uniform formal introduction of any type of lists. The presentation is open to other possible instantiations of *Position* parameter. In order to illustrate this, the case of unrolled linked lists is presented.

Also, this approach emphasizes the differences between the abstract data types of linked lists and the linked representation of the structures.

1. Introduction

Lists are very important data structures, which are widely used in computer science.

A recursive definition of an ADT for untyped, and mutable lists, specifies the domain as:

$$\mathit{List} = \{l | l \text{ is empty} \vee l = (e, l1), \text{ where } l1 : \mathit{List} \wedge e \text{ is an entity}\}$$

and the operations:

(1) a constructor for creating an empty list;
(2) an operation for testing whether or not a list is empty;
(3) an operation for prepending an entity to a list (cons in Lisp);
(4) an operation for determining the first component (or the "head") of a list (car in Lisp);

Received by the editors: March 10, 2006.

2000 *Mathematics Subject Classification.* 68P05, 68P01.

1998 *CR Categories and Descriptors.* E.1 [**Data**]: Data Structures – *Lists, stacks, and queues*.

(5) an operation for referring to the list consisting of all the components of a list except for its first (or its "tail") (cdr in Lisp);

This definition is preferred in functional languages, but in imperative programming, in which iteration is preferred to recursion, this definition is not so appropriate.

Under the imperative paradigm, a list is usually defined as an instance of an abstract data type (ADT) formalizing the concept of an ordered collection of entities. However, there are a lot of types of list data structures, and also in the literature could be found different definitions for ADT List. Also, the ADTs List are introduced very often in an informal way. Different implementations of list data structures are discussed more than a general ATD definition for them, and because of this there is not a uniform approach of the subject in the literature [2, 4, 5, 6, 7].

In practice, lists are usually implemented using arrays or linked lists of some sort; due to lists sharing certain properties with arrays and linked lists. Many times, the term list is used synonymously with linked list. *Sequence* is another used name, emphasizing the ordering and suggesting that it may not be a linked list. However, it is generally assumed that elements can be inserted into a list in constant time, while access of a random element in a list requires linear time; this is to be contrasted with an array (or vector), for which the time complexities are reversed.

Lists have the following properties:

- The contents or data type of lists may or may not vary at runtime.
- Lists may be typed. This implies that the entries in a list must have types that are compatible to the list base type.
- They may be sorted or unsorted.
- Random access over lists may or may not be possible.
- Equality of lists:
  : - In mathematics, sometimes equality of lists is defined simply in terms of object identity: two lists are equal if and only if they are the same object.
  : - In modern programming languages, equality of lists is normally defined in terms of structural equality of the corresponding entries, except that if the lists are typed, then the list types may also be relevant.

We propose, in this paper, a formal, general, non-recursive definition for a typed, unsorted ADT *List*. This could be used, then, as the starting point for a formal introduction of all the aspects referring to lists.

2. ADT List

We consider a list being a collection of elements of the same type (TE), where each element has a certain position. Each element in a list has a *successor* element and a *predecessor* element in the list, with two exceptions: the first element that has only a successor, and the last element that has only a predecessor.

The constructive approach is used for defining abstract data types.

We define a parameterized Abstract Data Type *List*, with two type parameters:

(i) The type parameter TE, which represents the type of the constitutive elements, characterized by at least two operations: *assign*, and *equals*.

(ii) *Position* is a type parameter that emphasizes the type of elements' positions. The instances of the type parameter *Position* are characterized by the existence of two operations: *next*, and *prev*. The properties of this type parameter are strictly connected to the type *List*. Knowing a position of an element in a list, we have, based on the list definition, to be able to extract the element stored at that position, and to compute successor and predecessor elements in the list.

Domain

$List(Position, TE) = \{l | l$ is a list of elements of type TE,
in which each element has a position of type $Position\}$

Operations:

(1) $createEmpty(l)$
pre: true
post: $l : List$ and l is empty

(2) $length(l)$
pre: $l : List$
post: $result =$ the number of the elements of the list l

(3) $getFirstPosition(l)$
pre: $l : List$
post: $result = \begin{cases} \perp, l \text{ is empty list} \\ p, p : Position \text{ is the first position in the non-empty list } l \end{cases}$

(4) $getLastPosition(l)$
pre: $l : List$
post: $result = \begin{cases} \perp, l \text{ is empty list} \\ p, p : Position \text{ is the last position in the non-empty list } l \end{cases}$

(5) $valid(l, p)$
pre: $l : List$ and $p : Position$
post: $result = \begin{cases} \text{true, if } p \text{ is a valid position in } l \\ \text{false, otherwise} \end{cases}$

(6) $addFirst(l, e)$
pre: $l : List$ and $e : TE$
post: $l' = (e, l)$

(7) $insert(l, p, e)$
pre: $l : List$ and $p : Position$ and $e : TE$ and $valid(p, l)$
post: l' is the list l after inserting e on the next position of p

(8) $delete(l, p)$
pre: $l : List$ and $p : Position$ and $valid(p, l)$
post: l' is the list l after removing the element on the position p

(9) $next(l, p)$
pre: $l : List$ and $p : Position$ and $valid(p, l)$
$result = \begin{cases} \text{the next position of } p\text{, if } p \text{ is not the last position} \\ \perp\text{, if } p \text{ is the last position} \end{cases}$

(10) $prev(l, p)$
pre: $l : List$ and $p : Position$ and $valid(p, l)$
post: $result = \begin{cases} \text{the previous position of } p\text{, if } p \text{ is not the first position} \\ \perp\text{, if } p \text{ is the first position} \end{cases}$

(11) $getElement(l, p)$
pre: $l : List$ and $p : Position$ and $valid(p, l)$
post: $result =$ the element e at the position p and $e : TE$

(12) $setElement(l, p, e)$
pre: $l : List$ and $p : Position$ and $valid(p, l)$ and $e : TE$
post: the element at the position p is equal to e

(13) $iterator(l)$
pre: $l : List$
post: $result =$ iterator of type *ListIterator* on the list l

We have used the notation $\perp$ for the undefined position, which is a special value of *Position* type. (The undefined position is always not valid, but a value which is not valid for a list does not have to be equal to $\perp$.) In the formal specification of an operation with parameter l, l' denotes l after the execution of that operation. We consider *ListIterator*, the type of a bidirectional "Read-Write" iterator on lists, characterized by the following operations (beside the creational operations):

(1) $valid(it, l)$
pre: $l : List$ and $it : ListIterator$
post: $result = \begin{cases} \text{true, if } it \text{ indicates no element in the list } l \\ \text{false, otherwise} \end{cases}$

(2) $next(it, l)$
pre: $l : List$ and $it : ListIterator$ and $valid(it, l)$
post: it' indicates the next position of that indicated by it

(3) $prev(it, l)$
pre: $l : List$ and $it : ListIterator$ and $valid(it, l)$
post: it' indicates the previous position of that indicated by it

(4) $element(it, l)$
pre: $l : List$ and $it : ListIterator$ and $valid(it, l)$
post: $result = e$, where e is the element indicated by it

(5) $insert(it, l, e)$
pre: $l : List$ and $it : ListIterator$ and $e : TE$ and $valid(it, l)$
post: l' is l after inserting e on the next position of that indicated by it

(6) $delete(it, l)$
pre: $l : List$ and $it : ListIterator$ and $valid(it, l)$
post: l' is l after removing the element on the position indicated by it

Since *insert* and *delete* operations are defined for *ListIterator*, a list could also be modified using an iterator. More restrictive iterator types could be considered, too.

The position of one element in the list allows us to obtain the element, and it may be given either by giving the index of the element in the list, or by given a reference to the location where that element is stored.
So, possible choices for *Position* are:

(1) $Index = \{i | i \in \mathbb{N}\}$
The resulted lists are characterized by the
ADT $List(Index, TE) = IndexedList(TE)$.

(2) $SNode = \{(e, n) | e : TE \wedge n : \mathrm{ref}(SNode)\}$
The resulted lists are characterized by the
ADT $List(\mathrm{ref}(SNode), TE) = SinglyLinkedList(TE)$.

(3) $DNode = \{(p, e, n) | e : TE \wedge n, p : \mathrm{ref}(DNode)\}$
The resulted lists are characterized by the
ADT $List(\mathrm{ref}(DNode), TE) = DoublyLinkedList(TE)$.

The notation ref($Type$) specifies a type of references to values of type $Type$. The types $ref(SNode)$ and $ref(DNode)$ use the value *null* (it could be a null pointer or a null reference to an entry into an array) for specifying $\perp$, and the type *Index* could use the value 0 for the same purpose.

By choosing concrete instances of type *Position* we obtain abstract data types, which are parameterized only by the elements' type TE. For an ADT like this, we may choose different representations of the values of the domain *List* and corresponding implementations of the operations. So, different list data structures are obtained.

Other instantiations for *Position* may be considered. For example, for a *unrolled linked* or *chunk* list that is a linked list in which each node contains an array of data values, the *Position* could be defined as a reference to the type

$$ArrayNode = \{(A, next, max, ind) | \\ A : Array(TE) \wedge next : \mathrm{ref}(ArrayNode) \wedge max, ind \in \mathbb{N}^*\}$$

where max represents the number of the elements stored in the node, and ind represents the current position into the node. The unrolled linked lists increase the cache performance while decreasing the memory overhead for references.

3. *IndexedList*

*IndexedList*s are characterized by the fact that elements are accessed directly by their indices. The internal representation of the lists is independent of their interface, so we may considere an array representation, but also a linked representation for *IndexedList*.

If we consider that the lists are represented using a dynamic array (its size is not constant) of elements of type TE, we obtain a list data structure in which direct access to the elements is possible in an $O(1)$ time, but *delete* and *insert* operations need each $O(n)$ time. Usually, this data structure is called *ArrayList* or *Sequence*.

Another possibility to implement *IndexedList* is based on a linked representation. In this case, each element is placed in a node that contains not only the value of the element, but also a pointer to the node that contains the next element in the list; in a doubly linked representation, a pointer to the node that contains the previous element in the list is stored, too. This representation of the *IndexedList* does not improve the time complexity of the operations *insert* and *delete*, since the positions are still referred to by indices. The improvement could be obtained if the insertions and deletions are done using an iterator, and not using the operations of the interface.

The majority languages defines in their collection libraries implementations of the ADT *IndexedList*. This is the case of Java [3], where the interface *List* corresponds to the interface of *IndexedList*, and the classes *ArrayList* and *LinkedList* are implementations of this. Therefore, the class *LinkedList* has not the type that corresponds to a linked list since its interface is an *IndexedList* interface. Only, its representation is a linked one. The same situation there is in C++, in STL library [8].

The *IndexedList* ADT is preferred to be implemented, since its interface does not contain parameters of type pointer or reference, which may bring some problems.

4. *LinkedList*

We will treat together the ADT *SinglyLinkedList* and ADT *DoublyLinkedList* since their problems are similar.

The positions, in this case, are expressed by the addresses where the elements are stored, and the elements are accessible through them. For representation, we may consider both static and dynamic linked allocation.

Dynamic linked allocation is based on nodes that contains the element's value and pointers to the node of the next element (and to the previous element, for the *DoublyLinkedList*). A representation for *SinglyLinkedList*s may use a pointer to the first node, and a representation for *DoublyLinkedList*s may use two pointers to the first and last nodes.

For *DoublyLinkedList*s, the *insert* and *delete* operations take $O(1)$ time, fact that corresponds to the general meaning of lists.
Another advantage of *DoublyLinkedList*s is that an easy reverse traversal is possible.

In the *SinglyLinkedList* case, *insert* operation also takes $O(1)$ time. But *delete* operation takes $O(n)$ time, because it has to compute the previous position of the element to be deleted, and the *prev* operation takes $O(n)$ time, in this case.

Static linked storage means that not dynamically allocation is used, but an associative array. For *SinglyLinkedList*s each entry of the associative array contains a value of type TE and a link to the sucessor element. Inside an array the reference(address) of an element is given by its index, so in this case a link is expressed as an index. The free space is also managed as a linked list. In the *DoublyLinkedList* case, an entry of the associative array also contains a link to the predecessor element.

5. Lists with Cursor

For each ADT List, a representation with a cursor could be chosen. In this case, a current position is stored (in the cursor) for any list, so the *Position* parameter p of the operations: *insert*, *delete*, *next*, *prev*, *getElement*, and *valid*, is included in the *List* parameter.

This representation is not common for indexed lists, but it brings some advantages for linked lists. The complexity of the operations is not influenced by the storing of this current position, but the advantage is given by the fact that the user has not direct access to the addresses where the elements are stored, and so the information is better protected. For linked lists, this cursor acts as an iterator on that list.

6. Conclusions

There are a lot of definitions of lists in literature, and they are very often informally defined. We have presented in this paper a formal definition for a general ADT List, from which specialized ADTs List are obtained by instantiation of the type parameter used for specifying positions in lists. The discussed ADTs, which are obtained after *Position* instantiation are *IndexedList*, *SinglyLinkedList*, and *DoublyLinkedList*. This analysis that starts from a general parameterized ADT has the advantage of a uniform formal introduction of any type of lists. The presentation is open to other possible instantiations of *Position* parameter.

This formal and general definition of lists emphasizes very clearly the differences between an indexed list (in which elements are referred to by using their number into the list) implemented in a linked way (as LinkedList in Java), and an implementation of ADT LinkedList.

Acknowledgment: This general and formal definition of lists was developed after long and interesting discussions that I had with Gabriela Serban referring to Data Structures Course. I am grateful for her suggestions and observations.

References

[1] Aho, A. V., Data Structures and Algorithms. Addison-Wesley, 1983.
[2] Amsbury,W., Data Structures - From Arrays to Priority Queues. 1985.
[3] Arnold, K., Holmes, D., Gosling, J., *The Java Programming Language*. Addison-Wesley, 2000.
[4] Horowitz, E, Sartaj Sahni, Fundamentals of Data Structures in C++. Computer Science Press, New York, 1995.
[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., Introduction to algorithms, sec. ed. MIT Press 2001.
[6] Mount, D. M., Data Structures, University of Maryland, 1993.
[7] Standish, T.A.,Data Structures, Algorithms and Software Principles. Addison-Wesley, 1994.
[8] Musser, D.R., Scine A., *STL Tutorial and Reference Guide: C++ Programming with Standard Template Library*, Addison-Wesley, 1995.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca
E-mail address: vniculescu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

A FRAMEWORK PROPOSAL FOR FINE GRAINED ACCESS CONTROL

COSTA CIPRIAN

Abstract. One of the main concerns in database security is confining a user to a specific set of the existing data. Current common DBMS implementations usually rely on specifying the columns a user might or might not see. The more intricate problem of restricting the user to several specified partitions of the data is usually left to the programmers and implemented through views (either simple or parameterized). This has long been acknowledged by the research community and the concept of query rewriting and "authorization transparent" queries has been introduced and discussed in great detail [1, 2, 4]. The scope of this paper is to introduce a framework for expressing the rules used in access control. The novelties introduced by this framework are: ability to choose between the type of access control (rewriting, checking), ability to use the structure of the query as input for the access control routines, ability to specify access control routines as a set of rules connecting specific users with specific constraints. The framework is part of the larger effort of defining AQL - a query language based on a code generation framework [5].

1. Introduction

The problem addressed by this paper is how we restrict the users of a database to specific sets of data. We have several answers to this problem and we will present some of them along with their advantages and disadvantages.

One of the solutions would be to create views that the users will use instead of the relations in the database. That means that there should be a view (simple or parameterized) for each user or group of users and that the client must be aware of which view to use for which user. Aside from that, the big draw-back here

Received by the editors: May 22, 2006.

1998 *CR Categories and Descriptors.* H.2.7 [**Information Systems**]: Database Management – *Database Administration - Security, integrity, and protection*; H.3.3 [**Information Systems**]: Information Storage and Retrieval – *Information Search and Retrieval - Information filtering* .

is maintainability: both changes of the underlying schema (database evolution) and changes in the security strategies (requirements evolution) result in all views needing to be rewritten.

A big advantage of using views is that they can introduce computed data as part of the schema. For example, a security restriction might be that the grade of a student should only be available as a logical typed column stating whether the student has passed or not.

A disadvantage of using views is that, in some cases, more relations need to be joined in order to compute the set of data the user is allowed to see. When a query uses several of these views, some of the tables might be joined several times needlessly, those consuming execution and optimizer time. The problem here is that when you define the view, you don't have information about the overall structure of the query - a problem that is addressed by our solution.

There are, of course, query rewriting algorithms that can do the job of replacing the relations with the corresponding views transparently for the user. This allows for the query writing and rights management issues to be orthogonal, and also might address some of the performance issues (since one of the primary usages of query rewriting is optimization). The disadvantages are that the views still have to be maintained and, more importantly, these features are not supported by commercial RDBMS's.

Another approach (that also takes the query rewriting path) is the Virtual Private Database feature of Oracle's 9iR2 RDBMS [2]. This feature allows for predicates to be added to the where clause of the query in which a relation appears. Although it allows for fine grained transparent access control and usage of data that is not in the original result set, it is still quite limited in terms of taking advantage of the query structure.

The approach introduced in [1] advocates that access control should not interfere with the expected behavior of a query since it is likely that it will produce unexpected results. The approach uses the notions of non-Truman models, authorization views and conditional validity in order to apply the declared restrictions. Although the inference rules proposed are capable of taking the query structure into consideration (to some extent), it still requires the creation and management of views and is not supported by commercial RDBMS's. Another difference of this approach is that it allows for transparent validation of queries but the query writing and access control are still tightly coupled and not at all orthogonal since no query rewriting takes place.

Our approach is based on representing the query as an XML document with a specified schema (defined in the AQL system). Although the system was first designed so that the user writes XML documents instead of queries, the XML document could just as well be generated by a translator based on the considered language's grammar.

Since our approach relies on modifying any aspect of the query, we are actually talking about a code generation framework, so we can also use here all the research on the topic of code generation as well [5, 6, 7].

2. AQL

For the purpose of self containment we will introduce here some issues regarding AQL, a declarative language that we developed for the broader topic of database evolution.

AQL is an instance of the code generation framework that we created and which is based on XML documents and transformations applied on those documents (either XSLT scripts or plug-ins written in any programming language).

From an architectural point of view AQL uses the compiler approach of multiple serial transformations. Each XML document contains tags that instruct the compiler which code generation strategy it should use. Once a strategy is identified, the steps from within that strategy are applied on the XML document and the result of the final step is the query statement.

This approach is similar to the one adopted in [3] and the main advantages over other generation techniques (string based approaches, translators) are the ease of evolution and separation of concerns.

We will continue with a simple example (a select statement). Keep in mind that even if the XML document we start with does not appear to be user friendly, it is created by either a graphical user interface or by an SQL parser. Let's consider the select statement which returns all the quantities that have been shipped from an OrderDetails relation. The starting document would then be:

```
<StrategyParam Name="SimpleSelect">
  <Extraction>
   <HeaderColumns>
     <ColumnRef Name="Quantity"></ColumnRef>
   </HeaderColumns>
   <Source SourceID="0">
     <Columns>
       <Column Name="Quantity" Type="num"/>
```

```
      <Column Name="IsShipped" Type="logical"/>
     </Columns>
     <Table>OrderDetails</Table>
   </Source>
   <Filters>
     <CreationFilter>
       <Expression>
         <ColumnRef Name="IsShipped"></ColumnRef>
       </Expression>
     </CreationFilter>
   </Filters>
 </Extraction>
</StrategyParam>
```

Through a series of transformations that will transform the filters, collapse header columns, check for computed fields, etc., the final result will be:

```
select Quantity from OrderDetails where IsShipped = 1
```

The transformation that makes it possible to use in a filter a single column of type bit (which is not possible in TransactSQL - the targeted SQL dialect) has the following elements:

(1) An XPath expression identifying all the nodes that require the transformation - in our case all the ColumnRef tags that are not children of an operator
(2) An XSLT transformation that creates the expression Column = 1
(3) A hint on what to do with the result of the previous statement - in our case the action would be to replace the input tag

This is just one of the simplest applications of AQL. For a more in-depth analysis of AQL and it's features please refer to [5].

3. Fine Grained Access Control Framework

The access control framework that we propose is based on AQL code generation framework. As depicted in the above example, each step of an AQL code generation strategy has access to entire query and to any level of detail, so it is possible to take the best decision regarding how the query should be rewritten.

3.1. **Access control scenarios.** In order to understand the complexity of the access control problem we will consider some of the scenarios that could become a reality in a large application.

The simplest and most common access control scenario is to restrict a user to only see some of the columns in a relation (this is the scenario that is taken into account by most of the commercial RDBMS's today). In our framework this is possible by removing all the top-level references to columns that are not visible to the user.

A derived scenario is when the user is not allowed to see the actual values in the columns but is allowed to use them for other operations (such as joins for example). If we want to implement this in a RDBMS we will need to create views or stored procedures that encapsulate the joins and then expose the result to the user. The big problem here is that the user will be confined to the set of joins that we create and this could be a serious drawback in open database schemas. In our framework this can be implemented by restricting the space in which the ColumnRef tags are searched to the Header and filter sections.

Another common access rule that involves columns is the restriction that certain columns cannot be used unless they are aggregated. For example a user might not be allowed to see the actual grade of a student, but might be allowed to see the average grade for the entire year.

The more complex access rules are the ones referring to the rows that a user is allowed to see. In most cases this translates in appending a predicate to the where clause and maybe join some other relations in order to get the required data. However, there are some hidden complexities for this scenario. For example, a user might want to find the difference between his grade and the average grade. The SQL statement from which they would start in AQL (this is not a correct SQL statement in most of the RDMS we are aware of) is:

```
select grade - avg(grade) from Grades
```

This sentence would be translated by AQL as

```
select grade - AVG_GRADE
from Grades, (select avg(Grade) as AVG_GRADE from Grades)
```

If the user is not allowed to see the grades of all the students (but, for example only for the student with the StudentID of 1), we must not apply this filter when computing the average, because, in that case, the difference would always be 0. So, the correct translation is:

```
select grade - AVG_GRADE
from Grades, (select avg(Grade) as AVG_GRADE from Grades)
where Grades.StudentID = 1
```

As you can see, it is not always enough to just add a predicate to the where statement. The rule in this case is: "if the query uses the Grades relation and uses a column outside of an aggregate function then append the StudentID = 1 predicate to the where statement".

3.2. **Framework description.** Recent work on rights management has identified the problem of access control checks as being a typical application of crosscutting concerns and there are many examples of how to use Aspect Oriented Programming to solve this. Although AQL is not a typical object oriented language, we still can apply some of the AOP techniques (at least the ones using the compile-time code generation approach).

In figure 1 we showed the intended usage of the framework within a complex deployment. Besides enabling access control, we also get rid of problems like sending SQL code from the client (which, even if not recommended, is very often used in middle sized applications).

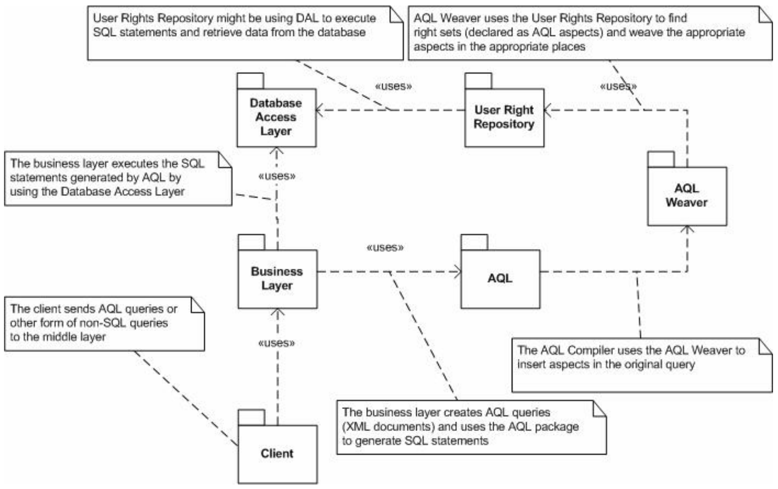

User Rights Repository might be using DAL to execute SQL statements and retrieve data from the database
AQL Weaver uses the User Rights Repository to find right sets (declared as AQL aspects) and weave the appropriate aspects in the appropriate places
«uses»
«uses»
Database Access Layer
User Right Repository
The business layer executes the SQL statements generated by AQL by using the Database Access Layer
«uses»
AQL Weaver
«uses»
Business Layer
AQL
The client sends AQL queries or other form of non-SQL queries to the middle layer
«uses»
The AQL Compiler uses the AQL Weaver to insert aspects in the original query
The business layer creates AQL queries (XML documents) and uses the AQL package to generate SQL statements
Client

FIGURE 1. Access Control Framework

One of the big advantages of using an XML based document to represent source code is the ability to execute complex queries on that source code. Many techniques used in refactoring are based on an intermediary, xml-based representation of the source code ([8]). The consequence in our case is that we can easily express elements of aspects by using an XML query language (in our case XPath).

An AQL Aspect is defined as AQLAspect = (Cond, JoinPoint, AspectCode) where:

- Cond is an XPath expression that evaluates to one or more XML nodes if the aspect is relevant for the AQL query. For example an horizontal expression on table A would have the condition ".//Table[@Name='A']" (which evaluates to one or more nodes if the table A is used in the query
- A join point is used to identify a set of points where the aspect code should be weaved. A JoinPoint is defined as JoinPoint = (BaseSelector, ExtensionSelector), where:
  - BaseSelector is an XPath expression that identifies the node which will be used to further identify the actual join point. In our previous example, the base selector should identify the select statement using table A: ".//Table[@Name='A']/parent::*"
  - ExtensionSelector is also an XPath expression that identifies the actual join point (relative to the node or nodes identified by the base selector). In our example the extension selector should identify the filters section of the select statement: "./Filters"
- AspectCode is an XML node that should be inserted at the join points previously identified. I our example it should specify a filter: "¡Filter¿Column = Value¡/Filter¿"

3.3. **Examples.** In order to better understand the usage scenarios of the framework and the extent of its capabilities, we created the following examples:

(1) Restrict access to the students of group 931. This could be done by adding a simple filter whenever the Students table is accessed.

```
A1 = (
".//Table[@Name='Students']",
(".//Table[@Name='Students']/parent::*", "./Filters"),
"<Filter> group='931' </Filter>")
```

(2) Restrict a teacher to the groups he has classes with. We will assume that the relation between Teacher and Groups is kept in a separate relation TeacherGroups. In order to do that we will need to join the students table to the TeacherGroups and Teachers table and then add a filter on the UserID column of the Teachers table.

```
A1 = (
".//Table[@Name='Students']",
(".//Table[@Name='Students']/parent::*", "."),
"<Join>
<Table Name=" TeacherGroups"/>
<Condition Value="Students.Group = TeacherGroups.Group"/>
 <Join>")

A2 = (
".//Table[@Name='Students']",
(".//Table[@Name='Students']/parent::*", "."),
"<Join>
<Table Name="Teachers"/>
<Condition Value="Teacher.TeacherID = TeacherGroups.TeacherID"/>
 <Join>")

A3 = (
".//Table[@Name='Students']",
(".//Table[@Name='Students']/parent::*", "./Filters"),
"<Filter> UserID = @UserID </Filter>")
```

In the second example we used the following aspects:

- A1 in order to insert a join to the TeacherGroups table
- A2 in order to insert a join to the Teachers table
- A3 in order to restrict the selection to the groups assigned to the current user

3.4. **Advantages and disadvantages.** The most obvious disadvantages to this approach are the fact that the user must use AQL and that performance might be negatively affected if each query is serialized in an XML document, transformations are applied and, at the end, the actual query is obtained.

The usage of AQL might not be a disadvantage since we are creating a complete solution that addresses ease of querying, schema and requirement evolution, access control, etc. However, if the system is already implemented and has to be translated to AQL just for the sake of access control, this is probably not the best solution.

As far as the algorithms are not implemented within the native query compiler and the targeted level of generality is high, there will always be a certain amount of performance penalties. In order to mitigate this, smart caches could be implemented, but only if the queries that are sent to the database are not very diverse.

There are also several advantages, some given by the fact that we are using AQL (we mentioned them earlier). However, as far as access control is concerned the biggest advantages are the fact that the entire query can be analyzed in order to decide how to restrict access and the fact that there is no limit on the type of transformations that are applied to the query.

Another issue that is broadly discussed by the research community is the fact that query rewriting will change the expected behavior of queries and might introduce very subtle problems in complex scenarios. This is true, but we consider that in the case of AQL there is such a big amount of query rewriting that the user must acknowledge and understand the inner workings of the rewriting algorithm, and, as such, is less likely to overlook important details.

4. Conclusions and further work

The framework that we present here has a very specific set of characteristics that set it apart from other frameworks that are available ([2], [3]). Because of this, the framework is going to address a particular type of applications, in which the access control patterns are very diverse and there are many clients writing queries and we want to implement security transparently, without affecting the number and complexity of the relations in the database. Because of this, we can identify two main directions in which research should continue:

(1) Defining complete, reusable design patterns for access control and implementing them in AQL. For this purpose we must consider a broad range of commercial applications and identify requirements, define best practices and implement them in a reusable and extensible fashion
(2) Continuing the development of AQL and especially improving its performance and range of available mechanisms. This is a very important task if this language is to be adopted in real-life scenarios and used in large applications.

References

[1] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, Prasan Roy, Shariq Rizvi: Extending Query Rewriting Techniques for Fine Grained Access Control, SIGMOD 2004 June 13-18, 2004, Paris, France.

[2] The Virtual Private Database in Oracle9ir2: An Oracle Technical White Paper http://otn.oracle.com/deploy/security/oracle9ir2/pdf/vpd9ir2twp.pdf.

[3] Galen S. Swint, Calton Pu, Gueyoung Jung, Wenchang Yan, Younggyun Koh, Qinyi Wu, Charles Consel, Akhil Sahai: Clearwater: Extensible, Flexible, Modular Code Generation, ASE'05, November 7-11, 2005, Long Beach, California, USA.

[4] A. Halevy. Answering queries using views: A survey. The VLDB Journal, 10(4):270-294, 2001.
[5] C. Costa: An XML evolution based approach to code generation, about to be published
[6] Alfred V. Aho, Mahadevan Ganapathi, Steven W. K. Tjiang: Code Generation Using Tree Matching and Dynamic Programming, ACM Transactions on Programming Languages and Systems, Vol. 11, No. 4, October 1989, Pages 491-516.
[7] Mahadevan Ganapathi, Charles N. Fischer: Affix Grammar Driven Code Generation, ACM Transactions on Programming Languages and Systems, Vol. 7, No. 4, October 1985. Pages 560-599.

Babes-Bolyai University, Faculty of Mathematics and Computer Science, Department of Computer Science, Cluj-Napoca, Romania

E-mail address: costa@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

CSÖRNYEI ZOLTÁN, *FORDÍTÓPROGRAMOK* (COMPILERS – IN HUNGARIAN), TYPOTEX BUDAPEST, 2006, PP 326, ISBN 963-9548-83-9

KÁSA ZOLTÁN

The contents of this book are based on the lectures held by the author for students at the Faculty of Informatics, Loránd Eötvös University in Budapest. The eleven chapters are: Introduction, Structure of a compiler, Parsers, Syntactical analyzers, Top-down analyzers, Bottom-up analyzers, Symbol table, Semantical analyzers, Error handling, Code generation, Code optimization.

The aim of the book is to present how a compiler works, how it detects the source errors, and how it prepares from the source code the excutable target code. A programmer who know the inside of a compiler and gets acquainted with translation algorithms will be able to write better and more efficient programs.

The book deals with the translation algorithms regarding the imperative languages. Theoretical backgrounds and efficient and practical methods used in compiler design and construction are in detail presented. Because the author focuses his interest on practical aspects, principles and translation methods, the proof of theorems are left to the reader's interest and can be found in referred papers. The book contains a lot of examples written in a easy readable pseudocode.

The book is good written, carefully printed, with 37 references and an exhaustive index table. You are invited to find more information about this and other interesting books on the web site of the publishing company, `http://www.typotex.hu`.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `kasa@cs.ubbcluj.ro`

Received by the editors: April 19, 2006.