Anul XLVII 2002

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 **XLVII**, Number 1, 2002

A NEW REAL TIME LEARNING ALGORITHM

GABRIELA ŞERBAN

Abstract. It is well known that all Artificial Intelligence problems require some sort of searching [7], that is why *search* has represents an important issue in the field of Artificial Intelligence. Search algorithms are useful for problem solving by intelligent (single or multiple) agents. In this paper we propose an original algorithm (RTL), which extends the Learning Real-Time A* (LRTA*) algorithm [1], used for solving path-finding problems. This algorithm preserves the completeness and the characteristic of LRTA* (a *real-time* search algorithm), providing a better exploration of the search space. Moreover, we design an Agent for solving a path-finding problem (searching a maze), using the RTL algorithm.
Keywords: Search, Agents, Learning.

1. Introduction

One class of problems addressed by search algorithms is the class of path-finding problems. Given a set of states (configurations), an initial state and a goal (final) state, the objective in a path-finding problem is to find a path (sequence of moves) from an initial configuration to a goal configuration.

In single-agent problem solving, the question is [7] that an agent is assumed to have *limited rationality*, so, the computational ability of an agent is usually limited. Therefore, the agent must do a limited amount of computations using only partial information on the problem.

The A* algorithm ([2]), a standard search algorithm, extends the wavefront of explored states from the initial state and chooses the most promising state within the whole wavefront. In this case, at each step, the global knowledge of the problem is required, that is why the computational complexity is considerable. So, the task is to solve the problem by accumulating local computations for each node in the graph (the search problem). These local computations can be executed concurrently (the execution order can be arbitrary), so, the problem could be solved both by single and multiple agents.

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*;

2. Path-Finding Problem

A path-finding problem consists of the following components [7]:

- a set of nodes, each representing a state;
- a set of directed links, each representing an operator available to a problem solving agent (each link is weighted with a positive number representing the cost of applying the operator - called *distance*);
- a unique node called the *start node*;
- a set of nodes, each of which represents a goal state.

We call the nodes that have directed links from node i *neighbors* of node i.

The problem is to find a path from the initial state to a goal state. In the followings we will refer to the problem of finding an optimal (shortest) path from the initial state to a goal state (we call the shortest path the path having the shortest distance to goal).

Notational conventions used in the followings are:

- *h(s)* - the shortest distance from node s to goal nodes;
- *h'(s)* - the estimated distance from node s to goal nodes;
- *k(s,s')* - the distance (cost of the link) between s and s'.

3. Learning Real-Time A*

When only one agent is solving a path-finding problem, it is not always possible to perform local computations for all nodes (for example, autonomous robots may not have enough time for planning and should interleave planning and execution). That is why the agent must selectively execute the computations for certain nodes. The problem is which node should choose the agent.

A way is to choose the current node were the agent is located. The agent updates the *h'* value of the current node, and then moves to the best neighboring node. This procedure is repeated until the agent reaches a goal state. The method is called the Learning Real-Time A* algorithm [1].

The algorithm is described in Figure 1.

---

(1) Calculate $f(j) = k(i,j) + h'(j)$ for each neighbor j of the current node i

(2) **Update:** Update the estimate of node i as follows:

$$h'(i) := min_j f(j) \tag{1}$$

(3) **Action selection:** Move to the neighbor j that has the minimum $f(j)$ value.

Figure 1. The Learning Real-Time A* algorithm.

One characteristic of the algorithm is that the agent determines the next action in a constant time. That is why this algorithm is called an *on-line*, *real-time* search algorithm.

The function that gives the initial values of h' is called a *heuristic function*. A heuristic function is called *admissible* if it never overestimates (in the worst case, the condition could be satisfied by setting all estimates to 0).

In LRTA*, the updating procedures are performed only for the nodes that the agent actually visits. The following characteristic is known [1]:

- In a finite number of nodes with positive link costs, in which there exists a path from every node to a goal node, and starting with non-negative admissible initial estimates, LRTA* is *complete*, i.e., it will eventually reach a goal node.

Since LRTA* never overestimates [7], it *learns* the optimal solution through repeated trials. In this case, the values learned by LRTA* will eventually converge to their actual distances along every optimal path to the goal node.

4. A Real-Time Learning Algorithm (RTL)

In fact, the behavior of the agent in the given environment can be seen as a Markov decision process. Regarding LRTA* there are two problems:

(1) in order to avoid recursion in cyclic graphs, it should be retained the nodes that have been already visited (with the corresponding values of *h'*). Therefore, the space complexity grows with the total number of states in the search space;
(2) what happens in some *plateau* situations - states in which, let us say, exists more successor (neighbor) states with the same minimum value for *h'* (the choice of the next action is nondeterministic).

In the followings, we propose an algorithm (RTL) which is an extension of the LRTA* algorithm, having some alternatives of solving the above presented problems. We mention that the algorithm preserves the completeness of LRTA*.

The proposed solutions for the problems (1) and (2) are:

(1) we keep a track of the visited nodes, but we do not retain the values of *h'* for each node;
(2) in order to choose the next action in a given state, the agent determines the set of states S (which were not visited by the agent) having a minimum value for *h'*. If S is empty, the training fails, otherwise, the agent chooses a random state from S as a successor state (this allows a better exploration of the search space).

The idea of the algorithm (based on LRTA*) is the following:

- through repeated trials (training episodes), the agent tries some paths (possible optimal) to a goal state, and retains the shortest one;

- the number of trials is selected by the user;
- after a training trial there are two possibilities:
  - the agent reaches a goal state; in this case the agent retains the path and it's cost;
  - the learning process fails (the agent does not reach the final state, because it was blocked).
- for avoiding cycles in the search space, the agent will not choose a state that was visited before, only if it has a single alternative (it was blocked) and it must return to the formerly visited state.

We make the following notations and assumptions:

- $S = \{s_1, \cdots, s_n\}$ - the set of states;
- $si \in S$ - the initial state;
- G - the set of goal states;
- $A = \{a_1, \cdots, a_m\}$ - the set of actions that could be executed by the agent;
- we assume that the state transitions are deterministic - a given action in a given state transitions to a single successor state (the Markov Model is not hidden [8]);
- with the former assumption, the transitions between states (and their costs') could be retained as a function $env : S\text{x}A\text{x}N \to S$ - if $s, s' \in S$, $a \in A$ and $c \in N$ so that if the agent takes the action a in the state s he reaches the state s' with the cost c, then $s' = env(s, a, c)$;
- we will say that the state s' is the *neighbor* of the state s iff $\exists a \in A$ and $c \in N$ so that $s' = env(s, a, c)$;
- *h'(s)* - the estimated distance from state s to a goal node;
- we will say that the *cost* of the path $s_1 \overset{a_1}{\to} s_2 \overset{a_2}{\to} \cdots \overset{a_{k-1}}{\to} s_k$ is $C = \sum_{i=1}^{k-1} c_i$, where $s_{i+1} = env(s_i, a_i, c_i)$ for all $i = 1, \cdots, k-1$.

The algorithm

The algorithm consists in a repeated update of the estimated values of the states, until the agent reaches a goal state (in fact a training sequence). The training is repeated for a given number of trials.

The algorithm is shown in Figure 2.

We have to mention that:

- we considered that if the agent finds in several trials the same optimal solution, then it is very probable that the solution is the correct one, and the training process stops;
- the time complexity (in the worst case) of the training process during one trial is $O(n^2)$, where n is the number of states of the environment;
- the agent determines the next action in a real-time (the selection process is a linear one);

Repeat until the number of trials was exceeded **or** until the correct solution was found

- **Training:**
  (1) **Initialization:**
    - sc (the current state):= si (the initial state)
    - calculate the estimation of the current state $h'(sc)$
  (2) **Iteration:**
  Repeat until ($sc \in G$) or (the agent was blocked) or (the number of visited states exceeds a maximum value)
    (a) **Update:**
      - **for** each state s' neighbor of sc the agent calculates the estimation of the shortest distance from s' to a goal state

$$f(s') = c + h'(s'), s' = env(sc, a, c) \tag{2}$$

      - the agent determines the set of states $M = \{s_1^{"}, \cdots s_k^{"}\}$ so that for all $j = 1, \cdots, k$

$$s_j^{"} = argmin_{s'}\{f(s') \mid \exists a \in A, c \in N \, so \, that \, s' = env(sc, a, c)\} \tag{3}$$

    (b) **Action selection:**
      - if $k = 1$ (the agent has a single alternative to continue) then the agent moves in the state $s_1^{"}$;
      - otherwise the agent determines from the set M a subset M' of states that were not visited in the current training sequence and chooses randomly a state from M'.

FIGURE 2. The Real-Time Learning (RTL) algorithm.

- the space complexity is reduced (there are retained only the states from the optimal path).

As in the LRTA* algorithm, if the heuristic function (the initial values of h') is *admissible* (never overestimates the true value -$h'(s)$ <= $h(s)$ for all $s \in S$-), then we can easily prove that the RTL algorithm is complete, i.e, it will eventually reach the goal [4] and $h'(s)$ will eventually converge to the true value $h(s)$ [6].

The proof of convergence is presented below:

Proof. Let $h'_n(s)$ be the estimation of the shortest distance from state s to a final state, at the n-th training episode. Let $h_n(s)$ be the shortest distance from state s to a final state, at the n-th training episode. Let $e_n(s) = h_n(s) - h'_n(s)$ be the estimation error at the n-th episode. We will prove that $lim_n e_n(s) = 0$, for all $s \in S$, which will assure the convergence of the algorithm.

Because h' never overestimates h it is obvious that

$$e_n(s) >= 0, for\, all\, n \in N, s \in S \tag{4}$$

From the updating step of the RTL algorithm (Figure 2) results that:

$$h'_n(s) = min_{s'}\{k(s,s') + h'_n(s')\}, \text{ s' neighbor of s} \tag{5}$$

for all $s \in S$, s visited by the agent in the current training sequence.

On the other hand, it is obvious that:

$$h_n(s) = min_{s'}\{k(s,s') + h_n(s')\}, \text{ s' neighbor of s} \tag{6}$$

for all $s \in S$.

Moreover, the real values of the shortest distance from a state s to goal are the same in all the training episodes, so that:

$$h_{n+1}(s) = h_n(s) \tag{7}$$

for all $s \in S$.

(8)

$h'_{n+1}(s) = h_n(s)$, if s was not visited, otherwise, $h'_{n+1}(s) = min_{s'}\{k(s,s')+h_n(s')\}$

From equations (7) and (8) results that:

$$e_{n+1}(s) - e_n(s) = h'_n(s) - h'_{n+1}(s) <= h'_n(s) - h'_n(s') <= 0 \tag{9}$$

where s' is neighbor of s and it is closer than s to a goal state (that is why it's estimation is less than the estimation of the current state).

From (4) and (9) results that $e_n(s)$ is convergent to 0. In other words, if the number of the training sequences is infinite, then the convergence of the algorithm is guaranteed.

5. An Agent for Maze Searching

5.1. **General Presentation.** The application is written in Borland C and implements the behavior of an Intelligent Agent (a robotic agent), whose purpose is coming out from a maze on the shortest path, using the algorithm described in the previous section (RTL).

We assume that:

- the maze has a rectangular form; in some positions there are obstacles; the agent starts in a given state and it tries to reach a final (goal) state, avoiding the obstacles;
- in a certain position on the maze the agent could move in four directions: north, south, east, west (there are four possible actions);
- the cost of executing an action (move in one direction) is 1;
- as a heuristic function (initial values for $h'(s)$) we have chosen the Manhattan distance to the goal (it is obvious that this heuristic function is admissible), which assures the completeness of the algorithm.

In fact it is a kind of semi-supervised learning, because the agent starts with an initial knowledge (the heuristic function) , so it has an informed behavior. In the worst case, if the values of the heuristic function are 0, then the learning is unsupervised, but the behavior of the agent becomes uninformed.

5.2. **The Agent's Design.** For implementing the algorithm, we will represent the following structures:

- a *State* from the environment;
- the *Environment* (as a linked list of *States*);
- a *Node* from the optimal path (the current *State* and the estimation *h'* of the current state);
- the optimal path from a training sequence (as a linked list of *Nodes*).

The basis classes used for implementing the agent's behavior are the followings:

- **IElement**: defines an interface for an element. This is an abstract class having two pure virtual methods:
  - for converting the member data of an element into a string;
  - a destructor for the member data.
- **CNode**: defines the structure of a *Node* from the optimal path. This class implements (inherits) the interface *IElement*, having (besides the methods from the interface) it's own methods for:
  - setting components (the current state, the estimation of the current state);
  - accessing components.
- **CState**: defines the structure of a *State* from the environment. This class implements (inherits) the interface *IElement*, having (besides the methods from the interface) it's own methods for:
  - setting components (the current position on the maze, the value of a state);
  - accessing components;
  - calculating the estimation *h'* of the state;
  - verifying if the state is accessible (contains or not an obstacle).
- **CList**: defines the structure of a linked list, with a generic element (a pointer to *IElement*) as information of the nodes. The main methods of the class are for:
  - adding elements;
  - accessing elements;
  - updating elements.
- **CEnvironment**: defines the structure of the agent's environment (it depends on the concrete problem - in our example the environment is a rectangular maze). The private member data of this class are:
  - **m**: the environment, represented as a linked list (*CList*) of states (*CState*);

  - **si**: the initial state of the agent (is a *CState*);
  - **sf**: the final state from the environment (is a *CState*);
  - **l**, **c**: the dimensions of the environment (number of rows and columns).

  The main methods of the class are for:
  - reading the environment from an input stream;
  - setting and accessing components;
  - verifying the neighborhood of two states in the environment.
- **Agent**: the main class of the application, which implements the agent's behavior and the learning algorithm.

  The private member data of this class are:
  - **m**: the agent's environment (is a *CEnvironment*);
  - **l**: the list of *Nodes* used for retaining the optimal path in the current training sequence (is a *CList*);

  The public methods of the agent are the followings:
  - **readEnvironment**: reads the information about the environment from an input stream ;
  - **writeEnvironment**: writes the information about the environment in an output stream ;
  - **learning**: is the main method of the agent; implements the RTL algorithm.

  Besides the public methods, the agent has some private methods used in the method **learning**.

We notice that all the representations of data structures are linked, which means that there are no limitations for the structures' length (number of states).

5.3. **Experimental Results.** For our experiment, we considered the environment shown in Figure 3. The state marked with 1 represents the initial state of the agent, the state marked with 2 represents the final state and the states filled with black contains obstacles (which the agent should avoid).

We repeat the experiment four times, because of the random character of the action selection mechanism. The results after the experiments are shown in Table 1, 2, 3, 4 (in a solution the agent determines the moving direction from the current state).

We notice that, in average, after 8 episodes, the agent finds the optimal path to the final state.

TABLE 1. First experiment

Number of episodes	8
The optimal solution	East North North East North North East East East North
Episode	Number of steps until the final state was reached
1	10
2	16
3	10
4	10
5	18
6	12
7	14
8	10

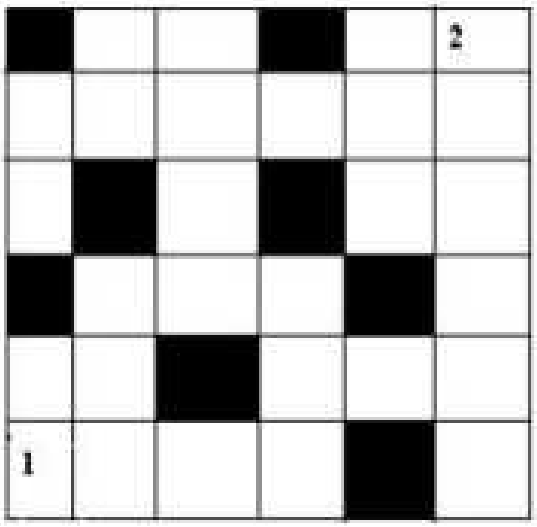

FIGURE 3. The agent's environment

6. Conclusions and Further Work

The algorithm described in this paper is very general, could be applied in any problem which goal is to find an optimal solution in a search space (a path-finding problem).

TABLE 2. Second experiment

Number of episodes	6
The optimal solution	East North North East North North East East East North
Episode	Number of steps until the final state was reached
1	16
2	10
3	14
4	10
5	10
6	10

TABLE 3. Third experiment

Number of episodes	14
The optimal solution	East North North East North North East East East North
Episode	Number of steps until the final state was reached
1	18
2	10
3	12
4	10
5	16
6	10
7	12
8	14
9	16
10	28
11	14
12	16
13	14
14	10

On the other hand, the application is designed in a way which allows us to model (with a few modifications) any environment and any behavior of an agent.

Further work is planned to be done in the following directions:

- to analyze what happens if the transitions between states are nondeterministic (the environment is a Hidden Markov Model [8]);
- to use probabilistic action selection mechanisms (ϵ-Greedy, SoftMax [5]);

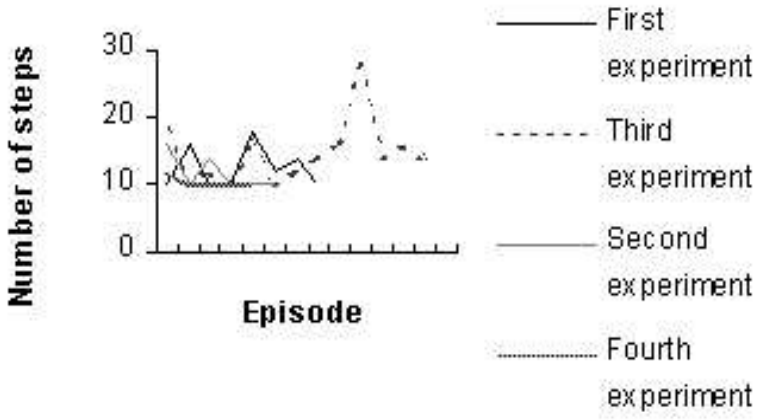

FIGURE 4. The number of steps/episode during the training processes

TABLE 4. Fourth experiment

Number of episodes	5
The optimal solution	East East East North East East North North North North
Episode	Number of steps until the final state was reached
1	12
2	10
3	12
4	10
5	10

- to combine the RTL algorithm with other classical path-finding algorithms (RTA*);
- in which way the agent could deduce the heuristic function from the interaction with it's environment (a kind of reinforcement learning);
- to develop the algorithm for solving path-finding problems with multiple agents.

REFERENCES

[1] Korf, R., E.: Real-time heuristic search. Artificial Intelligence, 1990
[2] Korf, R., E.: Search. Encyclopedia of Artificial Intelligence, Wiley-Interscience Publication, New York, 1992

[3] Russell, S.J., Norvig, P.: Artificial intelligence. A modern approach. Prentice-Hall International, 1995
[4] Ishida, T., Korf, R., E.: A moving target search. A real-time search for changing goals. IEEE Transaction on Pattern Analysis and Machine Intelligence, 1995
[5] Sutton, R., Barto, A., G.: Reinforcement learning. The MIT Press, Cambridge, England, 1998
[6] Shimbo, M., Ishida T.: On the convergence of real-time search. Journal of Japanese Society for Artificial Intelligence, 1998
[7] Weiss, G.: Multiagent systems - A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, Massachusetts, London, 1999
[8] Serban, G.: Training Hidden Markov Models - a Method for Training Intelligent Agents, Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, Krakow, Poland, 2001, pp.267-276

Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

A NEW ALGORITHM FOR SOLVING SELF-DUAL LINEAR OPTIMIZATION PROBLEMS

ZSOLT DARVAY

Abstract. Recently in [3] we have defined a new method for finding search directions for interior point methods (IPMs) in linear optimization (LO). Using one particular member of the new family of search directions we have developed a new primal-dual interior point algorithm for LO. We have proved that this short-update algorithm has also the $O(\sqrt{n}\log\frac{n}{\epsilon})$ iteration bound, like the standard primal-dual interior point algorithm. In this paper we describe a similar approach for self-dual LO problems. This method provides a starting interior feasible point for LO problems. We prove that the iteration bound is $O(\sqrt{n}\log\frac{n}{\epsilon})$ in this case too.

1. Introduction

In this paper we discuss polynomial methods for LO. The first polynomial algorithm for solving LO problems is the ellipsoid method of Khachiyan [6]. This method is important from a theoretical point of view, but is not so efficient in practice. An alternative variant was defined in 1984 by Karmarkar [5]. His projective method is the first IPM for LO. The field of IPMs has been very active since 1984. For an overview of results see the following books [1, 2, 10, 13, 14]. Let us consider the LO problem in canonical form

$$(P) \qquad \begin{aligned} &min\ c^T\xi \\ s.t.\quad &A\xi \geq b, \\ &\xi \geq 0, \end{aligned}$$

Received by the editors: 2002.05.30.

2000 *Mathematics Subject Classification.* 90C05.

1998 *CR Categories and Descriptors.* G.1.6. [**Mathematics of Computing**]: Numerical Analysis – *Linear programming*;

where $A \in \Re^{m\times k}$ with $rank(A) = m$, $b \in \Re^m$ and $c \in \Re^k$. The dual of this problem is:

$$(D) \qquad \begin{aligned} max\ & b^T\pi \\ s.t.\ \ & A^T\pi \le c, \\ & \pi \ge 0. \end{aligned}$$

It is well-known the following theorem.

Theorem 1.1 (strong duality) *Let $\xi \ge 0$ and $\pi \ge 0$ so that $A\xi \ge b$ and $A^T\pi \le c$, in other words ξ is feasible for (P) and π for (D). Then ξ and π are optimal if and only if $c^T\xi = b^T\pi$.* ■

This theorem implies that if (P) and (D) have optimal solutions then

$$(1) \qquad \begin{aligned} A\xi - z = b, &\quad \xi \ge 0, &\quad z \ge 0, \\ A^T\pi + w = c, &\quad \pi \ge 0, &\quad w \ge 0, \\ b^T\pi - c^T\xi = \rho, &\quad \rho \ge 0 \end{aligned}$$

has also a solution, where $z \in \Re^m$, $w \in \Re^k$ and $\rho \in \Re$ are slack variables. Furthermore, every solution of (1) provides optimal solutions of (P) and (D). Let us introduce the matrix $\bar{M}$ and the vectors $\bar{x}$ and $\bar{s}(\bar{x})$ as

$$\bar{M} = \begin{bmatrix} 0 & A & -b \\ -A^T & 0 & c \\ b^T & -c^T & 0 \end{bmatrix}, \quad \bar{x} = \begin{bmatrix} \pi \\ \xi \\ \tau \end{bmatrix}, \quad \text{and} \quad \bar{s}(\bar{x}) = \begin{bmatrix} z \\ w \\ \rho \end{bmatrix},$$

where $\tau \in \Re$. Consider the following homogeneous system

$$(2) \qquad \bar{s}(\bar{x}) = \bar{M}\bar{x}, \quad \bar{x} \ge 0, \quad \bar{s}(\bar{x}) \ge 0.$$

We mention that system (2) is the so-called *Goldman-Tucker model* [4, 12]. Let $\bar{n} = m + k + 1$ and observe that the matrix $\bar{M} \in \Re^{\bar{n}\times\bar{n}}$ is skew-symmetric, i.e. $\bar{M}^T = -\bar{M}$. Now we can state the following theorem.

Theorem 1.2 *Consider the primal-dual pair (P) and (D). Then we have*

1. *If ξ and π are optimal solutions of (P) and (D) respectively, then for $\tau = 1$ and $\rho = 0$ we obtain that $\bar{x}$ is a solution of (2).*
2. *If $\bar{x}$ is a solution of (2), then we have $\tau = 0$ or $\rho = 0$, thus we cannot have $\tau\rho > 0$.*
3. *If $\bar{x}$ is a solution of (2) and $\tau > 0$, then $(\frac{\xi}{\tau}, \frac{\pi}{\tau})$ is an optimal solution of the primal-dual pair (P)-(D).*
4. *If $\bar{x}$ is a solution of (2) and $\rho > 0$, then at least one of the problems (P) and (D) are infeasible.*

Proof: The first statement follows from the strong duality theorem. To prove the second one observe that

$$0 \leq \tau\rho = \tau b^T\pi - \tau c^T\xi = \pi^T(\tau b) - \tau c^T\xi = \pi^T A\xi - \pi^T z - \pi^T A\xi - \pi^T w \leq 0.$$

Thus $\tau\rho = 0$, and we get $\tau = 0$ or $\rho = 0$. Using this result the third assertion follows from Theorem 1.1. To prove the last statement suppose that both problems are feasible and $\rho > 0$. Thus there exists $\hat{\xi} \geq 0$ and $\hat{\pi} \geq 0$ so that $A\hat{\xi} \geq b$ and $A^T\hat{\pi} \leq c$. From $\rho > 0$ we get $\tau = 0$, therefore $A\xi \geq 0$ and $A^T\pi \leq 0$. Furthermore, from $\rho > 0$ we obtain that $b^T\pi > 0$ or $c^T\xi < 0$. If $b^T\pi > 0$ then

$$0 < b^T\pi \leq \hat{\xi}^T A^T\pi \leq 0,$$

and if $c^T\xi < 0$ then

$$0 > c^T\xi \geq \hat{\pi}^T A\xi \geq 0,$$

hence in both cases we have a contradiction. Thus the proof is complete. ■

In the next section we shall use the system (2) to accomplish the self-dual embedding of the primal-dual LO pair.

2. Self-Dual Embedding

In this section we investigate a generalized form of the system (2). Our approach follows the method proposed in [10]. Let us consider the LO problem

$$(\overline{SP}) \qquad \begin{array}{c} min\ \bar{q}^T\bar{x} \\ s.t. \quad \bar{M}\bar{x} \geq -\bar{q}, \\ \bar{x} \geq 0, \end{array}$$

where $\bar{M} \in \Re^{\bar{n}\times\bar{n}}$ is a skew-symmetric matrix, $\bar{q} \in \Re^{\bar{n}}$ and $\bar{q} \geq 0$. Moreover, let

$$\bar{s}(\bar{x}) = \bar{M}\bar{x} + \bar{q}.$$

We are going to solve $(\overline{SP})$ with an IPM, thus we need starting feasible solutions, so that $\bar{x} > 0$ and $\bar{s}(\bar{x}) > 0$. We say that in this case the problem $(\overline{SP})$ satisfies the *interior point condition* (IPC). Unfortunately such starting feasible solution for the problem $(\overline{SP})$ does not exist, but we can construct another problem equivalent to $(\overline{SP})$ which satisfies the IPC. For this purpose let

$$r = e - \bar{M}e \quad \text{and} \quad n = \bar{n} + 1,$$

where e denotes the all-one vector of length $\bar{n}$. Furthermore, introduce the notations

$$M = \begin{bmatrix} \bar{M} & r \\ -r^T & 0 \end{bmatrix}, \quad x = \begin{bmatrix} \bar{x} \\ \vartheta \end{bmatrix} \quad \text{and} \quad q = \begin{bmatrix} 0 \\ n \end{bmatrix},$$

and consider the problem

$$(SP) \qquad \begin{aligned} &min\ q^T x \\ &s.t. \quad Mx \geq -q, \\ &\qquad x \geq 0. \end{aligned}$$

Observe that the matrix M is also skew-symmetric, and problem (SP) satisfies the IPC. Indeed, we have

$$M \begin{bmatrix} e \\ 1 \end{bmatrix} + q = \begin{bmatrix} \bar{M} & r \\ -r^T & 0 \end{bmatrix} \begin{bmatrix} e \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ n \end{bmatrix} = \begin{bmatrix} \bar{M}e + r \\ -r^T e + n \end{bmatrix} = \begin{bmatrix} e \\ 1 \end{bmatrix}.$$

We have used that the matric $\bar{M}$ is skew-symmetric, thus $e^T \bar{M} e = 0$, and this equality yields

$$-r^T e + n = -(e - \bar{M}e)^T e + n = 1.$$

In order to solve the problem (SP) we use an IPM. Let

$$s = s(x) = Mx + q,$$

and consider the path of *analytic centers* [11], the primal-dual *central path*

$$(3) \qquad \begin{aligned} Mx + q &= s, \\ xs &= \mu e, \end{aligned}$$

where $\mu > 0$, and xs is the coordinatewise product of the vectors x and s, i.e.

$$xs = [x_1 s_1, x_2 s_2, \ldots, x_n s_n].$$

In fact for an arbitrary function f, and an arbitrary vector x we will use the notation

$$f(x) = [f(x_1), f(x_2), \ldots, f(x_n)]^T.$$

It is well-known that if the IPC holds for the problem (SP), then the system (3) has a unique solution for each $\mu > 0$. IPMs generally follow the central path by using Newton's method. In the next section we are going to formulate an equivalent form of the central path, and we shall apply Newton's method to obtain new search directions.

3. A New Class of Directions

New search directions have been studied recently by Peng, Roos and Terlaky [7, 9, 8]. In a recent paper [3] we have proposed a different approach for defining a new class of directions for LO. In this section we propose a similar approach for the self-dual problem (SP). Thus, we introduce a new class of directions for the problem (SP). Let $\Re^+ = \{x \in \Re \mid x \geq 0\}$, and let us consider the function

$$\varphi \in C^1, \qquad \varphi : \Re^+ \to \Re^+,$$

and suppose that the inverse function φ^{-1} exists. Then the system of equations which defines the central path (3) is equivalent to

$$\begin{aligned} Mx + q &= s, \\ \varphi\left(\frac{xs}{\mu}\right) &= \varphi(e). \end{aligned} \tag{4}$$

Using Newton's method for the system (4) we obtain new search directions for the problem (SP). Denote

$$v = \sqrt{\frac{xs}{\mu}},$$

and assume that $(x, s) > 0$ and $Mx + q = s$, thus x is an interior feasible solution of the problem (SP). Applying Newton's method for the system (4) we get

$$M\Delta x = \Delta s, \tag{5a}$$

$$\frac{s}{\mu}\varphi\prime\left(\frac{xs}{\mu}\right)\Delta x + \frac{x}{\mu}\varphi\prime\left(\frac{xs}{\mu}\right)\Delta s = \varphi(e) - \varphi\left(\frac{xs}{\mu}\right) \tag{5b}$$

We introduce the notations

$$d_x = \frac{v\Delta x}{x}, \qquad d_s = \frac{v\Delta s}{s}.$$

We have

$$\mu v(d_x + d_s) = s\Delta x + x\Delta s, \tag{6}$$

and

$$d_x d_s = \frac{\Delta x \Delta s}{\mu}. \tag{7}$$

Consequently (5b) can be written in the following form

$$d_x + d_s = p_v, \tag{8}$$

where

$$p_v = \frac{\varphi(e) - \varphi(v^2)}{v\varphi\prime(v^2)}.$$

Now using that M is skew-symmetric we get

$$\Delta x^T \Delta s = \Delta x^T M \Delta x = -\Delta x^T M \Delta x,$$

hence $\Delta x^T \Delta s = 0$. Moreover, from (7) follows

$$d_x^T d_s = e^T(d_x d_s) = \frac{1}{\mu}e^T(\Delta x \Delta s) = \frac{1}{\mu}\Delta x^T \Delta s = 0,$$

thus d_x and d_s are orthogonal. We shall use this relation later in the paper. We conclude that in this section we have defined a class of search directions for the problem (SP). For this purpose we have used a function φ to transform the

system (3) in an equivalent form. In the next section we shall consider a particular member of this class of search directions. Thus we shall develop a new polynomial algorithm for the self-dual problem (SP).

4. The Algorithm

In the remaining part of the paper we assume that $\varphi(x) = \sqrt{x}$. Using this function we present a new primal-dual interior-point algorithm for solving the problem (SP). Consequently, we obtain also a solution of (P) and (D). In this case applying Newton's method for the system (4) yields

$$\begin{aligned} M\Delta x &= \Delta s, \\ \sqrt{\frac{s}{\mu x}}\Delta x + \sqrt{\frac{x}{\mu s}}\Delta s &= 2\left(e - \sqrt{\frac{xs}{\mu}}\right). \end{aligned} \tag{9}$$

For $\varphi(x) = \sqrt{x}$ we have

$$p_v = 2(e - v), \tag{10}$$

and we can define a proximity measure to the central path by

$$\sigma(x, \mu) = \frac{\|p_v\|}{2} = \|e - v\| = \left\|e - \sqrt{\frac{xs}{\mu}}\right\|,$$

where $\|\cdot\|$ denotes the Euclidean norm (l_2 norm). Let us introduce the notation

$$q_v = d_x - d_s$$

Now using that the vectors d_x and d_s are orthogonal we obtain

$$\|p_v\| = \|q_v\|,$$

therefore the proximity measure can be written in the form

$$\sigma(x, \mu) = \frac{\|q_v\|}{2}.$$

Moreover, we have

$$d_x = \frac{p_v + q_v}{2}, \quad d_s = \frac{p_v - q_v}{2} \quad \text{and} \quad d_x d_s = \frac{p_v^2 - q_v^2}{4}. \tag{11}$$

The algorithm can be defined as follows.

Algorithm 4.1 *Let $\epsilon > 0$ be the accuracy parameter and $0 < \theta < 1$ the update parameter (default $\theta = \frac{1}{2\sqrt{n}}$).*

begin

$x := e$; $\mu := 1$;

while $n\mu > \epsilon$ **do begin**

$\mu := (1 - \theta)\mu$;

Compute Δx using (9);
$x := x + \Delta x$;
end
end.

In the next section we shall prove that this algorithm solves the linear optimization problem in polynomial time.

5. Complexity analysis

In this section we are going to prove that Algorithm 4.1 solves the problem (SP) in polynomial time. In the first lemma we investigate under which conditions the feasibility of the full Newton step is assured. Let $x_+ = x + \Delta x$ and

$$s_+ = s(x_+) = M(x + \Delta x) + q = s + M\Delta x = s + \Delta s.$$

Using these notations we can state the lemma.

Lemma 5.1 *Let $\sigma = \sigma(x, \mu) < 1$. Then the full Newton step is strictly feasible, hence $x_+ > 0$ and $s_+ > 0$.*

Proof: For each $0 \leq \alpha \leq 1$ introduce the notation $x_+(\alpha) = x + \alpha\Delta x$ and $s_+(\alpha) = s + \alpha\Delta s$. Then we have

$$x_+(\alpha)s_+(\alpha) = xs + \alpha(s\Delta x + x\Delta s) + \alpha^2\Delta x\Delta s,$$

and from (6) and (7) we obtain

$$\frac{1}{\mu}x_+(\alpha)s_+(\alpha) = v^2 + \alpha v(d_x + d_s) + \alpha^2 d_x d_s.$$

Furthermore, from (8) and (11) we get

$$\frac{1}{\mu}x_+(\alpha)s_+(\alpha) = (1-\alpha)v^2 + \alpha(v^2 + vp_v) + \alpha^2\left(\frac{p_v^2}{4} - \frac{q_v^2}{4}\right).$$

Using (10) we find that

$$v^2 + vp_v = 2v - v^2 = e - (e - v)^2 = e - \frac{p_v^2}{4},$$

and this relation leads to

$$\frac{1}{\mu}x_+(\alpha)s_+(\alpha) = (1-\alpha)v^2 + \alpha\left(e - (1-\alpha)\frac{p_v^2}{4} - \alpha\frac{q_v^2}{4}\right). \tag{12}$$

Evidently, the inequality $x_+(\alpha)s_+(\alpha) > 0$ is satisfied if

$$\left\|(1-\alpha)\frac{p_v^2}{4} + \alpha\frac{q_v^2}{4}\right\|_\infty < 1,$$

where $\|\cdot\|_\infty$ denotes the Chebychev norm (l_∞ norm). We have

$$\left\|(1-\alpha)\frac{p_v^2}{4}+\alpha\frac{q_v^2}{4}\right\|_\infty \le (1-\alpha)\frac{\|p_v^2\|_\infty}{4}+\alpha\frac{\|q_v^2\|_\infty}{4} \le$$

$$\le (1-\alpha)\frac{\|p_v\|^2}{4}+\alpha\frac{\|q_v\|^2}{4} = \sigma^2 < 1.$$

Hence, for each $0 \le \alpha \le 1$ we have $x_+(\alpha)s_+(\alpha) > 0$. Consequently, the sign of the continuous functions of α, $x_+(\alpha)$ and $s_+(\alpha)$ remains the same for every $0 \le \alpha \le 1$. Hence $x_+(0) = x > 0$ and $s_+(0) = s > 0$ yields $x_+(1) = x_+ > 0$ and $s_+(1) = s_+ > 0$. This completes the proof. ■

In the following lemma we formulate a condition which guarantees the quadratic convergence of the Newton process. We mention that in fact this requirement will be identical to that one used in Lemma 5.1, namely $\sigma(x,\mu) < 1$.

Lemma 5.2 *Let $\sigma = \sigma(x,\mu) < 1$. Then*

$$\sigma(x_+,\mu) \le \frac{\sigma^2}{1+\sqrt{1-\sigma^2}}.$$

Hence, the full Newton step is quadratically convergent.

Proof: We deduce from Lemma 5.1 that the full Newton step is strictly feasible, thus $x_+ > 0$ and $s_+ > 0$. Denote

$$v_+ = \sqrt{\frac{x_+s_+}{\mu}},$$

and observe that making the substitution $\alpha = 1$ in (12) that equation becomes

$$v_+^2 = e - \frac{q_v^2}{4}. \tag{13}$$

Thus

$$\min(v_+) = \sqrt{1-\frac{1}{4}\|q_v^2\|_\infty} \ge \sqrt{1-\frac{\|q_v\|^2}{4}} = \sqrt{1-\sigma^2}, \tag{14}$$

where for each vector ξ we denote $\min(\xi) = \min\{\xi_i \mid 1 \le i \le n\}$. Furthermore, (13) and (14) lead to

$$\sigma(x_+s_+,\mu) = \left\|\frac{e-v_+^2}{e+v_+}\right\| \le \frac{1}{1+\min(v_+)}\|e-v_+^2\| \le$$

$$\le \frac{1}{1+\sqrt{1-\sigma^2}}\left\|\frac{q_v^2}{4}\right\| \le \frac{1}{1+\sqrt{1-\sigma^2}}\frac{\|q_v\|^2}{4} = \frac{\sigma^2}{1+\sqrt{1-\sigma^2}}$$

Consequently, we have $\sigma(x_+s_+,\mu) < \sigma^2$, and this implies the lemma. ■

From the self-dual property of the problem (SP) follows that the duality gap is

$$2(q^Tx) = 2(x^Ts),$$

where x is a feasible solution of (SP), and $s = s(x)$ is the appropriate slack vector. For simplicity we also refer to x^Ts as the duality gap. In the following lemma we analyse the effect of the full Newton step on the duality gap.

Lemma 5.3 *Let $\sigma = \sigma(x,\mu)$ and introduce the vectors x_+ and s_+ such that $x_+ = x + \Delta x$ and $s_+ = s + \Delta s$. Then we have*

$$(x_+)^Ts_+ = \mu(n - \sigma^2).$$

Thus $(x_+)^Ts_+ \leq \mu n$.

Proof: Substituting $\alpha = 1$ in (12) results in

$$\frac{1}{\mu}x_+s_+ = e - \frac{q_v^2}{4},$$

and using this equation we get

$$(x_+)^Ts_+ = e^T(x_+s_+) = \mu(e^Te - \frac{e^Tq_v^2}{4}) = \mu(n - \frac{\|q_v\|^2}{4}) = \mu(n-\sigma^2)$$

This implies the lemma. ■

In the following lemma we investigate the effect on the proximity measure of a full Newton step followed by an update of the parameter μ. Assume that μ is reduced by the factor $(1-\theta)$ in each iteration.

Lemma 5.4 *Let $\sigma = \sigma(x,\mu) < 1$ and $\mu_+ = (1-\theta)\mu$, where $0 < \theta < 1$. We have*

$$\sigma(x_+,\mu_+) \leq \frac{\theta\sqrt{n} + \sigma^2}{1 - \theta + \sqrt{(1-\theta)(1-\sigma^2)}}.$$

Furthermore, if $\sigma < \frac{1}{2}$ and $\theta = \frac{1}{2\sqrt{n}}$ then $\sigma(x_+,\mu_+) < \frac{1}{2}$.

Proof: From (13) and (14) we deduce

$$\sigma(x_+,\mu_+) = \left\| e - \sqrt{\frac{x_+s_+}{\mu_+}} \right\| = \frac{1}{\sqrt{1-\theta}} \left\| \sqrt{1-\theta}e - v_+ \right\| =$$

$$= \frac{1}{\sqrt{1-\theta}} \left\| \frac{(1-\theta)e - v_+^2}{\sqrt{1-\theta}e + v_+} \right\| \leq \frac{1}{\sqrt{1-\theta}(\sqrt{1-\theta} + \min(v_+))} \left\| -\theta e + \frac{q_v^2}{4} \right\| \leq$$

$$\leq \frac{1}{1 - \theta + \sqrt{(1-\theta)(1-\sigma^2)}} \left(\theta\sqrt{n} + \left\| \frac{q_v^2}{4} \right\| \right) \leq \frac{\theta\sqrt{n} + \sigma^2}{1 - \theta + \sqrt{(1-\theta)(1-\sigma^2)}}.$$

Thus, the first part of the lemma is proved. Now observe that $n = m + k + 2 \geq 4$, hence for $\theta = \frac{1}{2\sqrt{n}}$ we get $1-\theta \geq \frac{3}{4}$. Consequently, from $\sigma < \frac{1}{2}$ follows $\sigma(x_+, \mu_+) < \frac{1}{2}$. Thus the proof is complete. ■

From Lemma 5.4 we conclude that the algorithm is well defined. Indeed, the requirements $x > 0$ and $\sigma(x, \mu) < \frac{1}{2}$ are maintained at each iteration. In the following lemma we discuss the question of the bound on the number of iterations.

Lemma 5.5 *Let x^k be the k-th iterate of Algorithm 4.1, and let $s^k = s(x^k)$ be the appropriate slack vector. Then $(x^k)^T s^k \leq \epsilon$ for*

$$k \geq \left\lceil \frac{1}{\theta} \log \frac{n}{\epsilon} \right\rceil.$$

Proof: Using Lemma 5.3 we find that

$$(x^k)^T s^k \leq \mu^k n = (1-\theta)^k \mu^0 n = (1-\theta)^k n,$$

thus the inequality $(x^k)^T s^k \leq \epsilon$ is satisfied if

$$(1-\theta)^k n \leq \epsilon.$$

Now taking logarithms, we may write

$$k \log(1-\theta) + \log(n) \leq \log \epsilon,$$

and using the equation $-\log(1-\theta) \geq \theta$ we observe that the above inequality holds if

$$k\theta \geq \log(n) - \log \epsilon = \log \frac{n}{\epsilon}.$$

Thus the proof is complete. ■

For $\theta = \frac{1}{2\sqrt{n}}$ we obtain the following theorem.

Theorem 5.6 *Let $\theta = \frac{1}{2\sqrt{n}}$. Then Algorithm 4.1 requires at most*

$$O\left(\sqrt{n} \log \frac{n}{\epsilon}\right)$$

iterations. ■

6. Concluding remarks

In this paper we have developed a new class of search directions for the self-dual linear optimization problem. For this purpose we have introduced a function φ, and we have used Newton's method to define new search directions. For $\varphi(x) = \sqrt{x}$ these results can be used to introduce a new primal-dual polynomial algorithm for solving (SP). We have proved that the complexity of this algorithm is $O\left(\sqrt{n}\log\frac{n}{\epsilon}\right)$.

References

[1] N. Andrei. *Mathematical Programming. Interior Point Methods*. Editura Tehnică, Bucureşti, 1999. (In Romanian).

[2] Zs. Darvay. *Interior Point Methods in Linear Programming*. ELTE, Budapest, 1997. (In Hungarian).

[3] Zs. Darvay. A new class of search directions for linear optimization. In *Proceedings of Abstracts, McMaster Optimizations Conference: Theory and Applications held at McMaster University Hamilton, Ontario, Canada*, page 18, August 1-3, 2002. Submitted to *European Journal of Operational Research*.

[4] A.J. Goldman and A.W. Tucker. *Theory of Linear Programming, Linear Inequalities and Related Systems*, volume 38 of *H.W. Kuhn and A.W. Tucker eds. Annals of Mathematical Studies*. Princeton University Press, Princeton, NJ, 1956.

[5] N.K. Karmarkar. A new polinomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[6] L.G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.

[7] J. Peng, C. Roos, and T. Terlaky. A new class of polynomial primal-dual methods for linear and semidefinite optimization. Technical Report TU Delft, NL–2628BLDelft, Faculty of Mathematics and Computer Science, The Netherlands, 1999.

[8] J. Peng, C. Roos, and T. Terlaky. Self-regular proximities and new search directions for linear and semidefinite optimization. Technical report, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2000.

[9] J. Peng, C. Roos, and T. Terlaky. A new and efficient large-update interior-point method for linear optimization. Technical report, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2001.

[10] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization. An Interior Approach*. John Wiley & Sons, Chichester, UK, 1997.

[11] Gy. Sonnevend. An "analytic center" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In A. Prékopa, J. Szelezsán, and B. Strazicky, editors, *System Modelling and Optimization: Proceedings of the 12th IFIP-Conference held in Budapest, Hungay, September 1985*, volume 84 of *Lecture Notes in Control and Information Sciences*, pages 866–876. Springer Verlag, Berlin, West-Germany, 1986.

[12] A.W. Tucker. *Dual systems of homogeneous linear relations*, volume 38 of *H.W. Kuhn and A.W. Tucker eds. Annals of Mathematical Studies*. Princeton University Press, Princeton, NJ, 1956.

[13] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, USA, 1997.

[14] Y. Ye. *Interior Point Algorithms, Theory and Analysis*. John Wiley & Sons, Chichester, UK, 1997.

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

E-mail address: darvay@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

A MODIFICATION OF THE CRAMER-SHOUP DIGITAL SIGNATURE SCHEME

CONSTANTIN POPESCU

Abstract. Digital signatures have been used in Internet applications to provide data authentication and non-repudiation services. Digital signatures will keep on playing an important role in future Internet applications. In this paper we propose a secure digital signature scheme based on the Strong RSA Assumption. Compared with the recent signature scheme by Cramer and Shoup, public keys in our scheme are a bit smaller but the two schemes have about the same computational efficiency.

Keywords: Signature schemes, efficiency, security, adaptively chosen message attack, hash functions

1. Introduction

In 1976 Diffie and Hellman [4] devised the concept of public key cryptography and showed that secret communication is possible without a prior exchange of a secret key, as was necessary previously. Their ingenious idea was to use two different keys, a public key for encryption and a private key for decryption. Based on this asymmetry, they further devised the concept of digital signatures. There are two most well-known public key cryptosystems, the RSA scheme and the ElGamal scheme, which can provide both digital signature and data encryption. In the following years, others realizations of digital signature schemes were proposed [2], [3], [9], [15], [17]. The RSA scheme [16] can be used to provide both digital signatures and public key encryption. Its security relies on the difficulty of factorizing a modulus which is the product of two large primes. The algorithms of ElGamal [5] can also provide digital signatures and public key encryption. These rely on the difficulty of finding discrete logarithms in the field of integers modulo a large prime p. Subsequent refinements have been made to the original ElGamal schemes, particularly to the signature scheme. For example, the Digital Signature Standard (DSS) algorithm [6] combines ElGamal signatures with a idea of Schnorr [17] to increase efficiency and provide short signatures. More recently, the Elliptic

2000 *Mathematics Subject Classification.* 94A60.
1998 *CR Categories and Descriptors.* E.3 [**Data**]: Data Encryption.

Curve Cryptosystems (ECC) [11], [12], [13], in which the difficulty of breaking the system is based on the difficulty of computing a discrete logarithm over an elliptic curve, has also been considered to become a standard in the IEEE P1363 project.

Since digital signature has one of the unique features associated with the public key cryptography, digital signature has been used in security services to provide non-repudiation services. For example, digital signature has been used in the Secure Electronic Transactions (SET) standard [18] to provide security of electronic transfers of credit and payment information over the Internet. Digital signature has been adopted by many security protocols, such as SSL [19], to provide data authentication and non-repudiation services.

In this paper we propose a digital signature scheme which is provably secure against adaptive chosen message attacks [9]. This improves on recent results by Gennaro et al. [8] in that we do not require that the involved hash function is division intractable. Compared with the recent signature scheme by Cramer and Shoup [3], public keys in our scheme are a bit smaller but the two schemes have about the same computational efficiency.

2. The Model of a Signature Scheme

A user's signature on a message m is a string which depends on m, on public and secret data specific to the user and, possibly on randomly chosen data, in such a way that anyone can check the validity of the signature by using public data only. The user's public data are called the *public key*, whereas his secret data are called the *secret key*. Obviously we would like to prevent the forgery of a user's signature without knowledge of his secret key. In this section we give a more precise definition of signature schemes and of the possible attacks against them.

Definition 1. *A digital signature scheme consists of three algorithms:*

***Gen**: On input of a security parameter 1^l this probabilistic algorithm output the signer's secret and public keys, x and y, respectively.*

***Sign**: On input of the signer's secret and public keys and a message $m \in \{0,1\}^*$ this algorithm outputs a signature σ on m.*

***Verify**: On input of a message m, a signature σ and the public key y of a signer, the algorithm **Verify** outputs true or false.*

A signature scheme must satisfy the following properties:

(1) **Correctness**: Signatures produced by the signer with **Sign** must be accepted by **Verify**.

(2) **Unforgeability**: A signature scheme must be existentially unforgeable under an chosen message attack. That is, we require that every attacker

has a negligible probability of success in the following game. The attacker is allowed to sequentially obtain signature on polynomially many messages of his choosing (i.e., messages are allowed to depend on signatures that the adversary has seen). He is then required to produce as output a message m for which he did not receive a signature and a second string σ. If $\mathbf{Verify}(m, \sigma, y) = true$ then the attacker is successful and, hence, the signature scheme is vulnerable to existential forgery.

There are two specific kinds of attacks against signature schemes: the *no-message attack* and the *known-message attack*. In the first scenario the attacker only knows the public key of the signer. In the second one the attacker has access to a list of message-signature pairs. According to the way this list was created, we distinguish four subclasses of known-message attacks:

(1) The *plain known-message attack*: the attacker has access to a list of signed messages, but he has not chosen them.
(2) The *generic chosen-message attack*: the attacker can choose the list of messages to be signed. However, this choice must be made before accessing the public key of the signer. We call attack generic because the choice is independent of the signer.
(3) The *oriented chosen-message attack*: as above, the attacker chooses the list of messages to be signed, but the choice is made once the public key of the signer has been obtained. This attack is oriented against a specific signer.
(4) The *adaptively chosen-message attack*: having knowledge of the public key of the signer, the attacker can ask the signer to sign any message that he wants. He can then adapt his queries according to previous message-signature pairs.

We now classify the expected results of an attack:

- Disclosing the secret key of the signer. It is the most serious attack. This attack is termed *total break*.
- Constructing an efficient algorithm which is able to sign any message. This is called *universal forgery*.
- Providing a new message-signature pair. This is called *existential forgery*. In many cases this attack is not dangerous, because the output message is likely to be meaningless. Nevertheless, a signature scheme which is not existentially unforgeable does not guarantee by itself the identity of the signer. For example, it cannot be used to certify randomly looking elements, such as keys.

Definition 2. *A signature scheme is secure if an existential forgery is computationally impossible, even under an adaptively chosen-message attack.*

The first secure signature scheme was proposed by Goldwasser et al. [10] in 1984.

3. Number Theoretic Assumptions

This section reviews some cryptographic assumptions necessary in the subsequent design of our signature scheme.

The Strong RSA Assumption was independently introduced by Baric and Pfitzmann [1] and by Fujisaki and Okamoto [7]. It strengthens the widely accepted RSA assumption that finding e^{th}-roots modulo n, where e is the public and thus fixed exponent, is hard to the assumption that finding an e^{th}-roots modulo n for any $e > 1$ is hard.

Definition 3 (Strong RSA Problem)**.** *Let $n = pq$ be an RSA-like modulus and let G be a cyclic subgroup of $\mathbb{Z}_n^*$ of order l_g. Given n and $z \in G$, the Strong RSA Problem consists of finding $u \in G$ and $e \in \mathbb{Z}_{>1}$ satisfying $z \equiv u^e (mod\ n)$.*

Assumption 1 (Strong RSA Assumption)**.** *There exists a probabilistic polynomial time algorithm K which on input 1^{l_g} outputs a pair (n, z) such that for all probabilistic polynomial-time algorithms P, the probability that P can solve the Strong RSA Problem is negligible.*

Consequently, if n is a safe RSA-modulus (i.e., $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' all prime), it is more cautions to work in the subgroup of quadratic residues modulo n, that is, in the cyclic subgroup $QR(n)$ generated by an element of order $p'q'$.

The next corollary shows that it is easy to find a generator g of $QR(n)$: it suffices to choose an element $a \in \mathbb{Z}_n^*$ satisfying $\gcd(a \pm 1, n) = 1$ and then to take $g = a^2\ mod\ n$. We then have $QR(n) =< g >$.

Proposition 1. *Let $n = pq$, where $p \neq q, p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' all prime. The order of the elements in $\mathbb{Z}_n^*$ are one of the set $\{1, 2, p', q', 2p', 2q', p'q', 2p'q'\}$. Moreover, the order of $a \in \mathbb{Z}_n^*$ is equal to $p'q'$ or $2p'q'$ if and only if $\gcd(a \pm 1, n) = 1$.*

Corollary 1. *Let $n = pq$, where $p \neq q, p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' all prime. Then, for any $a \in \mathbb{Z}_n^*$ such that $\gcd(a \pm 1, n) = 1$, $< a^2 > \subset \mathbb{Z}_n^*$ is a cyclic subgroup of order $p'q'$.*

The security of our digital signature scheme is based on the Strong RSA Assumption.

4. Our Secure Digital Signature Scheme

This section describes a secure digital signature scheme based on the Strong RSA Assumption. Let $\varepsilon > 1$ be a security parameter and let $l_p, l_{\lambda_1} > l_{\lambda_2}, l_{\gamma_1} > l_{\gamma_2}$ denote lengths. Define the integral ranges $\Lambda = \left[2^{l_{\lambda_1}} - 2^{l_{\lambda_2}}, 2^{l_{\lambda_1}} + 2^{l_{\lambda_2}}\right]$ and $\Gamma = \left[2^{l_{\gamma_1}} - 2^{l_{\gamma_2}}, 2^{l_{\gamma_1}} + 2^{l_{\gamma_2}}\right]$ such that for all $(x, e) \in \Lambda \times \Gamma$, we have $0 < x + 2^{2l_p} < e$. Finally, let $H : \{0,1\}^* \to \Lambda$ be a collision-resistant hash function [14].

4.1. **Key Generation.** To generate his public and secret key, a signer runs the following algorithm (**Gen**):

(1) Select random secret l_p-bit primes p', q' such that both $p = 2p' + 1$ and $q = 2q' + 1$ are also prime. Set the modulus $n = pq$.
(2) Chose two random elements $a, a_0 \in QR(n)$.
(3) The public key consists of the tuple (n, a, a_0, H) .
(4) The corresponding secret key consists of (p', q').

4.2. **Signature Generation.** To sign a message $m \in \{0,1\}^*$ a signer uses the following algorithm (**Sign**):

(1) Choose a prime $e \in \Gamma$ that was not used before.
(2) Choose a random integer $r \in \Lambda$.
(3) Compute $x = H(m \parallel e \parallel r)$ and $u = (a^x a_0)^{1/e} \pmod{n}$.
(4) Output the signature (u, e, r).

4.3. **Signature Verification.** Checking whether a tuple (u, e, r) is a valid signature on a message $m \in \{0,1\}^*$ with respect to the public key n can be done as follow (the algorithm **Verify**):

(1) Check whether $(u, e, r) \in \mathbb{Z}_n^* \times \Gamma \times \Lambda$.
(2) Compute $x' = H(m \parallel e \parallel r)$.
(3) Check whether $u^e \equiv a^{x'} a_0 \pmod{n}$.
(4) Output the signature true if none of the checks failed.

5. Efficiency and Security Analysis

The cost of the **Sign** algorithm can be broken down into three components:

(1) Generation of a random prime e from the interval $\left[2^{l_{\gamma_1}} - 2^{l_{\gamma_2}}, 2^{l_{\gamma_1}} + 2^{l_{\gamma_2}}\right]$.
(2) Computation of its inverse e^{-1}.
(3) Computation of u which requires 2 exponentiations: one full with e^{-1} and one small with x.

The cost of the last step can be reduced by amending **Gen** to generate a_0 as a power of a, i.e., choose a random $r' \in \Lambda$ and compute $a_0 = a^{r'}$. The value r' would then become part of the secret key. This amendment allows us to avoid the small exponentiation in the last step above, i.e., the signer would perform only one full exponentiation with the exponent $(x + r')\, e^{-1}$.

The only possible drawback is the potential loss in the range of a_0 since it is no longer generated independently from a. However, we note that a similar speedup was proposed by Cramer and Shoup [3] where it was claimed that, since, a is highly likely a generator of $QR(n)$, the distribution of the resultant public key does not change significantly.

The cost of signature verification in our scheme is dominated by step 3 which requires two exponentiations: one full to compute u^e and one small to compute $a^{x'}$. However, the verification equation can be changed to $u^e \left(a^{-1}\right)^{x'} \equiv a_0 \,(mod\ n)$ and hence the computation gets reduced to about one full exponentiation.

Next, we show that our signature scheme indeed satisfies the requirement for a secure signature scheme according to Definition 1. The correctness property follows from inspection of the scheme. It remains to prove the schemes security against an adaptively chosen message attack. Similar to [8] we require that:

- For every H a collision-resistant hash function, all primes $e \in \Gamma$ and every two messages m_1 and m_2 the distribution $H(m_1 \parallel e \parallel r)$ and $H(m_2 \parallel e \parallel r)$ induced be the random choice of r are statistically close.
- The Strong RSA Assumption holds in a world where there exists an oracle that on input a message m, a prime $e \in \Gamma$ and an $x \in \Lambda$ outputs an $r \in \Lambda$ such that $x = H(m \parallel e \parallel r)$.

Theorem 1. *The signature scheme presented above is secure against adaptively chosen messages attack under the Strong RSA Assumption and the further assumption that there exists a family of hash functions $\{\mathcal{H}\}$ satisfying the above requirements.*

Proof. Assume that the attacker A queries signature for K messages and then outputs a signature (u', e') on the message m'. We now show that if we take control over the hash function, then we can use this attacker to break the Strong RSA Assumption, i.e., we are given a z and an n and must find an w and v such that $w^v \equiv z \,(mod\ n)$.

Let $((u_1, e_1, r_1), m_1), ..., ((u_K, e_K, r_K), m_K)$ denote the signature-message pairs that are constructed during the interaction with A. In order for A to be successful its output $((u', e', r'), m')$ must satisfy $(u', e') \neq (u_i, e_i)$ for $1 \leq i \leq K$. Depending of whether $e_i \nmid e'$ for some i, there are two games to calculate a pair $(w, v) \in \mathbb{Z}_n^* \times \mathbb{Z}_{>1}$ satisfying $w^v \equiv z \ (mod\, n)$ from which we randomly chose one

each time then play with the attacker. As mentioned before, we are assuming that there is an oracle that input a message m, a prime $e \in \Gamma$ and an $x \in \Lambda$ outputs an $r \in \Lambda$ such that $x = H(m \parallel e \parallel r)$. The adversary is allowed to query this oracle as well. The first of the two game goes as follows:

(1) Select $x_1, ..., x_K \in \Lambda$ and $e_1, ..., e_K \in \Gamma$.
(2) Set $a = z^{\prod_{1 \leq l \leq K} e_l} \ mod\ n$.
(3) Choose a random $r \in \{0,1\}^{2l_p}$ and set $a_0 = a^r \ mod\ n$.
(4) For all $1 \leq i \leq K$, compute $u_i = z^{(x_i + r) \prod_{1 \leq l \leq K; l \neq i} e_l} \ mod\ n$.
(5) Start A, feed it the (u_i, e_i, r_i), where we get r_i from the oracle, and eventually obtain $\left(x'; \left[u' = \left(a^{x'} a_0\right)^{1/e'} \ mod\ n, e', r'\right]\right)$ with $x', r' \in \Lambda$ and $e' \in \Gamma$.
(6) If $\gcd(e', e_j) \neq 1$ for all $1 \leq j \leq K$ output fail and stop. Otherwise, let $\widetilde{e} = (x' + r) \prod_{1 \leq l \leq K} e_l$. Since $\gcd(e', e_j) = 1$ for all $1 \leq j \leq K$, we have $\gcd(e', \widetilde{e}) = \gcd(e', (x' + r))$. Hence, by the extended Euclidean algorithm, there exist $\alpha, \beta \in \mathbb{Z}$ such that $\alpha e' + \beta \widetilde{e} = \gcd(e', (x' + r))$. Therefore, letting $w = z^{\alpha} (u')^{\beta} \ mod\ n$ and $v = e' / \gcd(e', (x' + r)) > 1$ since $e' > (x' + r)$ we have $w^v \equiv z \ (mod\, n)$.

The previous game is only successful if A returns a new signature with $\gcd(e', e_j) = 1$ for all $1 \leq j \leq K$. We now present a game that solves the Strong RSA Problem in the other case, that is, when $\gcd(e', e_j) \neq 1$ for some $1 \leq j \leq K$. Note that $\gcd(e', e_j) \neq 1$ means $\gcd(e', e_j) = e_j$ since e_j is prime.

(1) Select $x_1, ..., x_K \in \Lambda$ and $e_1, ..., e_K \in \Gamma$.
(2) Choose a random $j \in \{1, ..., K\}$ and set $a = z^{\prod_{1 \leq l \leq K; l \neq j} e_l} \ mod\ n$.
(3) Choose a random $r \in \{0,1\}^{2l_p}$ and set $u_j = a^r \ mod\ n$ and $a_0 = u_j^{e_j} / a^{x_j} \ mod\ n$.
(4) For all $1 \leq i \leq K$, $i \neq j$, compute $u_i = z^{(x_i + e_j r - x_j) \prod_{1 \leq l \leq K; l \neq i, j} e_l} \ mod\ n$.
(5) Start A, feed it the (u_i, e_i, r_i), where we get r_i from the oracle, and eventually obtain $\left(x'; \left[u' = \left(a^{x'} a_0\right)^{1/e'} \ mod\ n, e', r'\right]\right)$ with $x', r' \in \Lambda$ and $e' \in \Gamma$.
(6) If $\gcd(e', e_j) \neq e_j$ output fail and stop. Otherwise, we have $e' = te_j$ for some t and can define $b = (u')^t / u_j \ mod\ n$ if $x' \geq x_j$ and $b = u_j / (u')^t \ mod\ n$ otherwise. Hence $b \equiv \left(a^{|x' - x_j|}\right)^{1/e_j} \equiv \left(z^{|\widetilde{e}|}\right)^{1/e_j} \ (mod\, n)$ with $\widetilde{e} = (x' - x_j) \prod_{1 \leq l \leq K; l \neq j} e_l$. Since $\gcd\left(e_j, \prod_{1 \leq l \leq K; l \neq j} e_l\right) = 1$ it follows that $\gcd(e_j, |\widetilde{e}|) =$

$\gcd(e_j, |x' - x_j|)$. Hence, by the extended Euclidean algorithm, there exist $\alpha, \beta \in \mathbb{Z}$ such that $\alpha e_j + \beta |\widetilde{e}| = \gcd(e_j, |x' - x_j|)$. Therefore, letting $u = z^{\alpha} b^{\beta} \ mod \ n$ and $e = e_j / \gcd(e_j, |x' - x_j|) > 1$ since $e_j > |x' - x_j|$, we have $u^e \equiv z \ (mod\, n)$.

Consequently, by playing randomly one of game 1 or game 2 with A one can solve the Strong RSA Problem. Since the latter is assumed to be infeasible, it follows that no such attacker can exist. ■

We now compare our signature scheme with some recent results. The scheme due to Gennaro et al. [8] is simpler and seemingly more efficient than our scheme. The scheme is simpler since it appears as a true hash-and-sign scheme very close to RSA. It uses a similar variation of the Strong RSA Assumption for the proof of security as we do. However, their requirements for a suitable hash function are non-standard, e.g., it is required to be division intractable.

An interesting sidenote is that the only practical realization of a suitable hash function presented in [8] is the so-called chameleon hashing which outputs primes. This yields a signature scheme that requires the signer to generate a random looking prime. The cost of signing thus becomes roughly the same as in our present scheme: generation of a large prime, computation of its inverse and a single exponentiation. The cost of verification is one exponentiation plus the cost of a message hash which is quite expensive due to the special hash function used.

Comparing our signature scheme to the one by Cramer and Shoup, we find that are similar in many aspects of security properties and associated costs. The public key size in our scheme is somewhat smaller than its counterpart in Cramer and Shoup. The latter consists of a tuple (n, h, x, e'), where n is a modulus, h and x are elements of $QR(n)$ and e' is a prime. In contrast, our scheme's public key is a tuple (n, a, a_0, H), where n is a modulus, a and a_0 are elements of $QR(n)$ and H is a hash function which is, incidentally, also needed in a Cramer and Shoup public key. Thus, the size difference is due to the prime e' in the latter. A Cramer-Shoup signature is a tuple (y, y', e) where e is a small prime, $y' \in QR(n)$ and $y \in \mathbb{Z}_n^*$, i.e., both are n-bit integers. This is about the same as for our scheme. The cost of signing in [3] amounts to generating a prime, computing its inverse and three exponentiations of which two are small (each with an exponent from the range of the underlying hash function) and one is full. Hence, the cost of signing is somewhat higher in [3] than in our scheme. Signature verification in the Cramer and Shoup translates into two small exponentiations which is a bit more efficient than in our scheme.

6. Conclusion

In this paper we proposed a digital signature scheme which is provably secure against adaptive chosen message attacks. Compared with the recent signature scheme by Cramer and Shoup [3], public keys in our scheme are a bit smaller but the two schemes have about the same computational efficiency.

References

[1] N. Baric, B. Pfitzmann, Collision-free accumulators and fail-stop signature schemes without trees, In Advances in Cryptology-EUROCRYPT'97, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, pp. 480-494, 1997.

[2] R. Cramer, I. Damgaard, New generation of secure and practical RSA-based signatures, In Advances in Cryptology-Crypto'96, pp. 173-185, 1996.

[3] R. Cramer, V. Shoup, Signature Schemes Based on the Strong RSA Assumption, IBM Research Report RZ 3083, 1998.

[4] W. Diffie, M. Hellman, New Directions in Cryptography, IEEE Transaction Information Theory, IT-22, 6, pp. 644-654, 1976.

[5] T. ElGamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transaction Information Theory, IT-31, 4, pp. 469-472, 1985.

[6] FIPS 186, Digital Signature Standard, US Department of Commerce/NIST, 1994.

[7] E. Fujisaki, T. Okamoto, A practical and provably secure scheme for publicly verifiable secret sharing and its applications, In Advances in Cryptology-EUROCRYPT'98, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, pp. 32-46, 1998.

[8] R. Gennaro, S. Halevi, T. Rabin, Secure hash-and-sign signatures without the random oracle, Proc. of Eurocrypt'99, LNCS, Springer-Verlag, 1999.

[9] S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, SIAM Journal on Computing, 17(2), pp. 281-308, 1988.

[10] S. Goldwasser, S. Micali, R. Rivest, A "Paradoxical" solution to the signature problem, Proc. of the 25th FOCS, pp. 441-448, 1984.

[11] N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation, 48, pp. 203-209, 1987.

[12] N. Koblitz, CM-Curves with Good Cryptographic Properties, Proceedings of Crypto'91, 1992.

[13] V. Miller, Uses of elliptic curves in cryptography, Advances in Cryptology, Proceedings of Crypto'85, Lecture Notes in Computer Sciences, 218, Springer-Verlag, pp. 417-426, 1986.

[14] National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS Publication 180-1, April 1995.

[15] C. Popescu, Signature Schemes Based on the Discrete Logarithm Problem, Anal. of University of Oradea, pp. 25-32, 2001..

[16] R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, Communications of the ACM, 21, pp. 120-126, 1978.

[17] C.P. Schnorr, Efficient Signature Generation by Smart Cards, Journal of Cryptology, Vol. 4, No. 3, pp. 161-174, 1991.

[18] SET Specification, http://www.visa.com/cgi-bin/vee/ht/ecomm/set/downloads.

[19] The Secure Sockets Layer Protocol, http://www.netscape.com/info/security-doc.html.

University of Oradea, Department of Mathematics, Str. Armatei Romane 5, Oradea, Romania

E-mail address: cpopescu@math.uoradea.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

BL-ALGEBRA STRUCTURE OF RGB MODEL

DAN NOJE AND MEDA TODOR

Abstract. The aim of this paper is to construct a BL-Algebra structure over RGB Model of colours. First we determin a method to obtain $t-$norms on RGB Model and their associate residuum from $t-$norms on unit interval and their associate residuum. These triangular norms and their associate residuums are used to construct the BL-Algebra structure over RGB model.

Keywords: RGB Model, T-Norms, BL-Algebras, Many-valued logic

Introduction

In a recent paper [6] V. Loia and S. Sessa studied the compression and decompression of gray scale images using fuzzy relations in the Basic Logic over $[0, 1]$.

In this paper we develop an algebraic structure on RGB model of colours [5] that will lead us to consider compression and decompression of colour images. Therefore, we consider that a colour of RGB model is an element of the set

$$RGB = \{(r, g, b) \mid r, g, b \in [0, 2^n - 1]\},$$

where n represents the number of bits on which one colour component is stored in computer's memory.

In RGB set, r, g and b represents the red, green and respectively blue component of the colour. For more details about RGB set see [7].

To construct the BL-Algebra over RGB we have to determin the t-norm and its associate residuum over RGB.

We have to remind now, some notions used later.

A *triangular norm* t on real unit interval (for short $t-norm$) [1], [3] is a binary operation,

$$t : [0, 1]^2 \to [0, 1],$$

such that t is commutative, associative, non-decreasing in both arguments and $t(0, x) = 0$, $t(x, 1) = x$ for any $x \in [0, 1]$. For brevity, we put $t(x, y) = xty$ for all $x, y \in [0, 1]$.

The *residuum* of the t-norm t [2], [3] is a unique operation $x \to_t y$ defined as

$$(x \to_t y) = \max\{z \in [0, 1] \mid xtz \leq y\}$$

2000 *Mathematics Subject Classification.* 03B50, 03G10, 08A70, 08A72.

1998 *CR Categories and Descriptors.* I.3.2 [**Computing Methodologies**]: Computer Graphics – *Graphic Systems.*

such that $xtz \leq y$ if and only if $z \leq (x \to_t y)$.

The following are the most famous t-norms used in fuzzy logic with the associate residuum:

- Lukasiewicz t-norm

$$xty = \max\{0, x + y - 1\},$$

$$(x \to_t y) = \min\{1, 1 - x + y\}.$$

- Gödel t-norm

$$xty = \min\{x, y\},$$

$$(x \to_t y) = \begin{cases} 1 \text{ if } x \leq y \\ y \text{ if } x > y \end{cases}.$$

- Goguen t-norm

$$xty = x \cdot y,$$

$$(x \to_t y) = \begin{cases} 1 \text{ if } x \leq y \\ y \text{ if } x > y \end{cases}.$$

The following properties hold:

$$x \wedge y = xt(x \to_t y); \tag{0.1}$$

$$(x \to_t y) \vee (y \to_t x) = 1. \tag{0.2}$$

Following [4], a $BL - Algebra$ $L = (L, \wedge, \vee, \star, \to, 0, 1)$ is an algebra with four binary operations such that for all $x, y, z \in L$:

1. $(L, \wedge, \vee, 0, 1)$ is a bounded distributive lattice;
2. $(L, \star, 1)$ is a commutative semigroup having 1 as unit element;
3. $x \wedge y = x \star (x \to y)$ (divisibility);
4. $z \leq (x \to y)$ if and only if $x \star z \leq y$ (residuation because of 3., i.e. $(x \to y) = \vee\{z \in L \mid x \star z \leq y\}$);
5. $(x \to y) \vee (y \to x) = 1$ (prelinearity).

The properties (0.1), (0.2) guarantee that $L_t = ([0, 1], \min, \max, t, \to_t, 0, 1)$ are important examples of BL-algebras over $[0, 1]$, where t is a continuous t-norm with its residuum $\to_t$.

1. T-Norms defined on RGB set

In this section we will develop a method to obtain from each t-norm defined on unit interval $[0,1]$, a t-norm defined on RGB set.

We start with the construction of the t-norm and its associate residuum on one component of RGB set. Applying this t-norm and its associate residuum on each component we determin the t-norm and its associate residuum on RGB set.

Let $[0,b]$ interval, where $b=2^n-1$, be the set of values of one component of RGB set.

Let us consider a one to one onto mapping $f:[0,1]\to[0,b]$, defined as follows:

$$f(x)=b\cdot x,$$

for any $x\in[0,1]$.

It is easy to see that its inverse mapping is $f^{-1}:[0,b]\to[0,1]$, defined as follows:

$$f^{-1}(x)=\frac{x}{b},$$

for any $x\in[0,b]$.

Let us consider a t-norm t and a binary operation $T:[0,b]^2\to[0,b]$, defined for any $x,y\in[0,b]$ as follows:

$$xTy=f\left(f^{-1}(x)\,t\,f^{-1}(y)\right). \tag{1.1}$$

Lemma 1.1. *The binary operation T defined as above is a t-norm defined on $[0,b]$ interval.*

Proof. It is easy to see that $f^{-1}(x)\in[0,1]$ for any $x\in[0,b]$. It follows that

$$f^{-1}(x)\,t\,f^{-1}(y)\in[0,1]$$

and fulfills all the properties of a $t-$norms.

Since f is a linear mapping it follows that

$$f\left(f^{-1}(x)\,t\,f^{-1}(y)\right)\in[0,b]$$

and fullfills all the properties of a $t-$norm.

Therefore T is a $t-$norm on $[0,b]$ interval. ■

In what follows we will develop a similar method to obtain the associate residuum of the $t-$norm defined on $[0,b]$ interval.

The residuum of the $t-$norm T defined on $[0,b]$ interval is the unique operation defined for all $x,\ y\in[0,b]$ as follows:

$$x\to_T y=\max\{z\in[0,b]\mid xTz\le y\}.$$

Lemma 1.2. *Let T be a $t-$norm defined on $[0,b]$ interval. The binary operation defined for all $x,\ y\in[0,b]$ as follows:*

$$x\to_T y=f\left(f^{-1}(x)\to_t f^{-1}(y)\right) \tag{1.2}$$

is the residuum of $t-$norm T, where t represents the $t-$norm defined on unit interval from which we have obtained the $t-$norm T as in (1.1).

Proof. Let $x,\ y \in [0,b]$. Since f and f^{-1} are non-decreasing functions we have:

$$x \to_T y = \max\{z \in [0,b] \mid xTz \leq y\}$$

$$= f\left(\max\left\{u \in [0,1] \mid f^{-1}(x)\, tu \leq f^{-1}(y)\right\}\right)$$

$$= f\left(f^{-1}(x) \to_t f^{-1}(y)\right).$$

This completes the proof. ■

The equations (1.1) and (1.2) are used now to determine the form of the most famous $t-$norms and of their associate residuums.

Example 1.3. Lukasiewicz type $t-$norm defined on $[0,b]$ interval:

$$xTy = f\left(\max\left\{0, f^{-1}(x) + f^{-1}(y) - 1\right\}\right)$$

$$= b \cdot \max\left\{0, \frac{x}{b} + \frac{y}{b} - 1\right\}$$

$$= \max\{0, x + y - b\};$$

$$x \to_T y = f\left(\min\left\{1, 1 - f^{-1}(x) + f^{-1}(y)\right\}\right)$$

$$= b \cdot \min\left\{1, 1 - \frac{x}{b} + \frac{y}{b}\right\}$$

$$= \min\{b, b - x + y\}.$$

Example 1.4. Gödel type $t-$norm defined on $[0,b]$ interval:

$$xTy = f\left(\min\left\{f^{-1}(x), f^{-1}(y)\right\}\right)$$

$$= b \cdot \min\left\{\frac{x}{b}, \frac{y}{b}\right\}$$

$$= \min\{x, y\};$$

$$x \to_T y = f\left(f^{-1}(x) \to_t f^{-1}(y)\right)$$

Since f^{-1} is not a decreasing function it follows that:

(i) if $x \leq y \Rightarrow f^{-1}(x) \leq f^{-1}(y)$. Therefor we get:

$$f^{-1}(x) \to_t f^{-1}(y) = 1.$$

Using equation (1.2) we obtain:

$$x \to_T y = f\left(f^{-1}(x) \to_t f^{-1}(y)\right)$$

$$= f(1) = b.$$

(ii) if $x > y \Rightarrow f^{-1}(x) > f^{-1}(y)$. Therefor we get:

$$f^{-1}(x) \to_t f^{-1}(y) = f^{-1}(y).$$

Applying equation (1.2) we obtain:

$$x \to_T y = f\left(f^{-1}(x) \to_t f^{-1}(y)\right)$$

$$= f\left(f^{-1}(y)\right) = y.$$

¿From (i) and (ii) it follows that:

$$x \to_T y = \begin{cases} b & \text{if } x \leq y \\ y & \text{if x} > y \end{cases}.$$

Example 1.5. Goguen type $t-$norm defined on $[0, b]$ interval:

$$xTy = f\left(f^{-1}(x) \cdot f^{-1}(y)\right)$$

$$= b \cdot \left(\frac{x}{b} \cdot \frac{y}{b}\right)$$

$$= \frac{x \cdot y}{b};$$

The residuum is obtained in the same way as it was obtained the residuum of Gödel $t-$norm. It has the following definition:

$$x \to_T y = \begin{cases} b & \text{if } x \leq y \\ y & \text{if x} > y \end{cases}.$$

Once we have determined the $t-$norm and its associate residuum on $[0, b]$ interval, we can determin the $t-$norm and its associate residuum on RGB set.

The $t-$norm T is defined on $[0, b]$ interval, which represents one component of RGB set. Therefore if we apply the $t-$norm T on each component of RGB set we obtain the $t-$norm defined on it as follows:

Definition 1.6. For all $x, y \in RGB$, where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, we have:

$$xT_{RGB}y = (x_1Ty_1, x_2Ty_2, x_3Ty_3,)$$

where T represents the $t-$norm defined on $[0, b]$ interval as in (1.1).

We are able to determin now the associate residuum of $t-$norm T_{RGB} defined on RGB set. Applying the residuum $\to_T$ on each component of RGB set we obtain the associate residuum of $t-$norm T_{RGB} defined as follows:

Definition 1.7. For all $x, y \in RGB$, where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, we have:

$$x \to_{T_{RGB}} y = (x_1 \to_T y_1, x_2 \to_T y_2, x_3 \to_T y_3,)$$

where $\to_T$ represents the associate residuum of $t-$norm T defined on $[0, b]$ interval as in (1.2).

2. The BL-algebra over RGB set

Once determined the $t-$norm T_{RGB} and its associate residuum $\to_{T_{RGB}}$ we can go further to develop a BL-algebra over RGB set.

First it is necessary to define the supremum and the infimum of two elements of RGB set. In what follows we will follow the same steps as in the case of $t-$norm T_{RGB}.

We start with the construction of these two binary operations on $[0, b]$ interval, which represents one component of RGB set, using the operations defined on unit interval.

Let $\vee$ be the supremum and $\wedge$ the infimum of two elements defined on unit interval. Then we define the supremum $\vee_{[0,b]}$ and the infimum $\wedge_{[0,b]}$ of any two elements x, y from $[0, b]$ interval as follows:

$$x \vee_{[0,b]} y = f\left(f^{-1}(x) \vee f^{-1}(y)\right); \tag{2.1}$$

$$x \wedge_{[0,b]} y = f\left(f^{-1}(x) \wedge f^{-1}(y)\right); \tag{2.2}$$

where f and f^{-1} are the functions defined in the previous section.

Applying $\vee_{[0,b]}$ and $\wedge_{[0,b]}$ on each component of RGB set, we obtain the supremum and the infimum on RGB set defined as follows:

Definition 2.1. For all $x, y \in RGB$, where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, we have:

$$x \vee_{RGB} y = \left(x_1 \vee_{[0,b]} y_1, x_2 \vee_{[0,b]} y_2, x_3 \vee_{[0,b]} y_3\right); \tag{2.3}$$

$$x \wedge_{RGB} y = \left(x_1 \wedge_{[0,b]} y_1, x_2 \wedge_{[0,b]} y_2, x_3 \wedge_{[0,b]} y_3\right); \tag{2.4}$$

where $\vee_{[0,b]}$ and $\wedge_{[0,b]}$ represents the supremum and the infimum defined on $[0, b]$ interval as in (2.1) and (2.2).

In what follows we will check some properties of supremum and infimum defined on RGB set that will be used later.

Lemma 2.2. *For any $x, y \in RGB$, where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, the following properties hold:*

$$x \wedge_{RGB} y = xT_{RGB}\left(x \to_{T_{RGB}} y\right); \tag{2.5}$$

$$(x \to_{T_{RGB}} y) \vee_{RGB} (y \to_{T_{RGB}} x) = 1_{RGB}, \tag{2.6}$$

where $1_{RGB} = (b, b, b)$.

Proof. It is enough to prove that the above properties hold on each component of RGB set.

To prove first property, we use equation (2.2) and we obtain:

$$x_1 \wedge_{[0,b]} y_1 = f\left(f^{-1}(x) \wedge f^{-1}(y)\right),$$

and from equation (0.1) it follows:

$$x_1 \wedge_{[0,b]} y_1 = f\left(f^{-1}(x)\, t\left(f^{-1}(x) \to_t f^{-1}(y)\right)\right).$$

Using now the equations (1.1) and (1.2) we get:

$$x_1 \wedge_{[0,b]} y_1 = f\left(f^{-1}(x_1)\, t f^{-1}\left(f\left(f^{-1}(x_1) \to_t f^{-1}(y_1)\right)\right)\right)$$

$$= f\left(f^{-1}(x_1)\, t f^{-1}(x_1 \to_T y_1)\right)$$

$$= x_1 T (x_1 \to_T y_1).$$

For the proof of the second property, by (1.2) and (2.1) we have:

$$(x_1 \to_T y_1) \vee_{[0,b]} (y_1 \to_T x_1) = f\left(f^{-1}(x_1 \to_T y_1) \vee f^{-1}(y_1 \to_T x_1)\right)$$

$$= f\left(f^{-1}\left(f\left(f^{-1}(x_1) \to_t f^{-1}(y_1)\right)\right) \vee f^{-1}\left(f\left(f^{-1}(y_1) \to_t f^{-1}(x_1)\right)\right)\right)$$

$$= f\left(\left(f^{-1}(x_1) \to_t f^{-1}(y_1)\right) \vee \left(f^{-1}(y_1) \to_t f^{-1}(x_1)\right)\right).$$

Using the equation (0.2) we get:

$$(x_1 \to_T y_1) \vee_{[0,b]} (y_1 \to_T x_1) = f(1) = b.$$

This completes the proof. ■

In what follows we introduce the BL-algebra structure over RGB set.

Let $0_{RGB} = (0, 0, 0)$ and $1_{RGB} = (b, b, b)$ be two constants from RGB set, then

Lemma 2.3. *The structure* $(RGB, \wedge_{RGB}, \vee_{RGB}, T_{RGB}, \to_{T_{RGB}}, 0_{RGB}, 1_{RGB})$ *is a BL-algebra.*

Proof. To prove that $(RGB, \wedge_{RGB}, \vee_{RGB}, T_{RGB}, \to_{T_{RGB}}, 0_{RGB}, 1_{RGB})$ is a BL-algebra, we have to show that the following properties hold:

(i) $(RGB, \wedge_{RGB}, \vee_{RGB}, 0_{RGB}, 1_{RGB})$ is a bounded distributive lattice;

(ii) $(RGB, T_{RGB}, 1_{RGB})$ is a commutative semigroup having 1_{RGB} as unit element;

(iii) $x \wedge y = x T_{RGB} (x \to_{T_{RGB}} y)$ (divisibility);

(iv) $z \leq (x \to_{T_{RGB}} y)$ iff $x T_{RGB} z \leq y$;

(v) $(x \to_{T_{RGB}} y) \vee_{RGB} (y \to_{T_{RGB}} x) = 1_{RGB}$.

The properties (i) and (ii) are obvious. The properties (iii) and (v) are proved in Lemma 2.2. The property (iv) follows from the definition of the associate residuum of the $t-$norm t defined on unit interval and from the mode in which we have obtained the $t-$norm T_{RGB} and its associate residuum $\to_{T_{RGB}}$ from it. ■

Once defined a BL-algebra structure on RGB model we can start to use it in image compression and decompression, but this is the subject of a next paper.

References

[1] Dumitrescu, D., Lazzerini, B., Jain, L.C. (2000) "Fuzzy Sets and their Application to Clustering and Training", Boca Raton, Fl.
[2] Hajek, P. (1998) "Basic Fuzzy Logic and BL-algebras", Soft Computing, Vol. 2, pp. 124-128.
[3] Klement, E.P., Mesiar, R., Pap, E. (2000) "Triangular Norms", Kluwer Academic Publishers, Dordrecht, The Netherlands.
[4] Hajek, P. (1998) "Metamathematics of Fuzzy Logic", Kluwer Academic Publishers, Dordrecht, The Nederlands.
[5] Ionescu, F. (2000) "Grafica in realitatea virtuala", Editura Tehnica, Bucuresti.
[6] Loia, V., Sessa, S. (2002) "Compression and Decompression of Fuzzy Relations in the Basic Logic over [0,1], (submitted).
[7] Noje, D., Bede, B. (2001)"MV-Algebra structure of RGB Model", Studia Informatica, Cluj-Napoca.

Department of Mathematics, Faculty of Science, University of Oradea,Armatei Romane 5, 3700 Oradea, Romania
E-mail address: dnoje@uoradea.ro

Department of Mathematics and IT, Faculty of Medicine and Pharmacy, University of Oradea, Armatei Romane 5, 3700 Oradea, Romania
E-mail address: medas@rdsor.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

DARR – A THEOREM PROVER FOR CONSTRAINED AND RATIONAL DEFAULT LOGICS

MIHAIELA LUPEA

Abstract. Default logics represent an important class of the nonmonotonic formalisms. Using simple by powerful inference rules, called *defaults*, these logic systems model reasoning patterns of the form "in the absence of information to the contrary of...", and thus formalize the default reasoning, a special type of nonmonotonic reasoning. In this paper we propose an automated system, called *DARR*, with two components: a propositional theorem prover and a theorem prover for constrained and rational propositional default logics. A modified version of semantic tableaux method is used to implement the propositional prover. Also, this theorem proving method is adapted for computing extensions because one of its purpose is to produce models, and extensions are models of the world described by default theories.

1. Introduction

One of the first formalizations of nonmonotonic reasoning was *classical default logic*, proposed by Reiter [6]. This logic system is based on first-order logic and introduces a new kind of inference rules called *defaults*. Defaults are used to draw conclusions by making implicit assumptions in the absence of information. Default logic is nonmonotonic because conclusions derived can be later invalidated by adding new information.

A *default theory* (D,W) consists of W, which is a set of consistent formulas of first-order logic (the facts) and a set of default rules D. The formulas of W are the axioms of the theory and a default rule has the form[1]:$d = \frac{\alpha:\beta}{\gamma}$, where $\alpha, \beta, \gamma$are formulas of first order logic,α is the *prerequisite* (Precond(d)) of the default d, β is the *justification* (Justif(d)) of the default d and γ is the *consequent* (Conseq(d)) of the default d.

In the paper we will use the notations: Justif(D)=$\bigcup_{d\in D}$ Justif(d), Prereq(D)= $\bigcup_{d\in D}$ Prereq(d), Concl(D)= $\bigcup_{d\in D}$ Concl(d).

1991 *Mathematics Subject Classification.* 03B70, 68T27, 68T37.

1998 *CR Categories and Descriptors.* I2 [**Artificial Intelligence**] Deduction and Theorem Proving – nonmonotonic reasoning and belief revision.

[1]Due to the (semi) representability results for these versions of default logic, we use in this paper only defaults with at most one justification (unitary default theories).

Informally, an extension for a default theory is a set of formulas derived from W using the standard inference rules of classical logic and the defaults. Formulas belonging to an extension are called *nonmonotonic theorems*, that means default conclusions of the default theory, which are not necessarily true, only plausible. A default theory may have zero, one or more classical extensions. The set of defaults used in the construction of an extension is called the *set of generating defaults* for the considered extension.

A default $d = \frac{\alpha:\beta}{\gamma}$ can be applied and thus derive γ if α is believed and *it is consistent to assumed* β.

Different variants (justified, constrained, rational) of default logic try to provide an appropriate definition of consistency condition for the justifications of the defaults, and thus to obtain many interesting and useful formal properties for these logic systems.

There are three computational problems specific to default logics:

Search problem: finding the extensions of a default theory.
Decision problems:

(1) deciding whether a formula belongs to at least one extension of a default theory (credulous perspective of the default reasoning);
(2) deciding whether a formula belongs to all extensions of a default theory (skeptical perspective of the default reasoning).

Automated theorem proving for default logics has began with solving the decision and searching problems for particular default theories: normal [6], ordered seminormal, and then was extended to general theories. The well known classical theorem proving methods: resolution, semantic tableaux method, connection method, were incorporated and adapted in the automated systems for default logics to solve specific tasks.

We will enumerate some of the automated reasoning system for default logics:

- **DeReS** [2] computes classical extensions for stratified default theories, using a semantic tableaux propositional prover.
- **Exten** [1] is based on an operational approach for computing classical, justified and constrained extensions.
- **GADEL** [5] uses the principles of genetic algorithms for computing classical extensions.
- **Xray** [9] represents an approach of the query-answering problem in constrained and cumulative default logics.

The aim of this paper is to introduce theoretical aspects regarding theorem proving in constrained and rational default logics and to describe an automated system implemented in C++, called *DARR*, for these variants of default logic.

2. Constrained and rational default logics

Constrained default logic was introduced by Schaub [7]. The consistency condition is a global one and it is based on the observation that in commonsense reasoning we assume facts, we memorize our assumptions and we verify that they do not contradict each other. The *actual extension* is embedded in a consistent *context* where are retained all the assumptions (justifications) used in the reasoning process.

Due to the global consistency condition, the constrained logic is strong regular, semi-monotonic, strongly commits to assumptions and guarantees the existence of extensions.

Theorem 2.1 [8]: Let (D,W) be a default theory and let E, C be sets of formulas. (E=actual extension,C=context) is a *constrained extension* of (D,W) if and only if:

E=Th(W ∪ Conseq(D')) and C=Th(W ∪Justif(D') ∪ Conseq(D'))
for a maximal set D'⊆D such that D' is grounded in W and W∪Justif(D')∪Conseq(D') is consistent.

This theorem states that the reasoning process formalized by constrained default logic is guided by a consistent context generated by a strong regular set of defaults. We observe that a default theory has always a constrained extension because D'=∅ is grounded in W, W is consistent and thus ∅ can be a set of generating defaults.

Rational default logic was developed in [4] as a version of classical default logic for solve the problem of handling disjunctive information. The property of rational default logic is that defaults with mutually inconsistent justifications are never used together in constructing an extension of a default theory.

This logic system is strongly regular but does not guarantee the existence of extensions, is not semi-monotonic and does not commit to assumptions.

Theorem 2.2: Let (D,W) be a default theory and let E and C be sets of formulas. (E=actual extension, C=context) is a *rational extension* of (D,W) if and only if:

E=Th(W ∪ Conseq(D')) and C=Th(W ∪ Justif(D') ∪ Conseq(D'))
for a maximal D'⊆D such that D' is grounded in W and are satisfied the following conditions:

(i) W ∪ Concl(D') ∪ Justif(D') is consistent

(ii)$\forall d \in$ D\D' we have: W ∪ Concl(D') ∪ {¬Precond(d)} is consistent **or**
W ∪ Concl(D') ∪ Justif(D' ∪{d}) is inconsistent

This theorem provides a necessary and sufficient criteria for the existence of a set of generating defaults of a rational extension. If condition (i) is satisfied by a set D', but condition (ii) is not satisfied, D' cannot be a set of generating defaults for a rational extension.

Proof: For proving theorem 2.2 we will use the original definition of a rational extension.

Definition 2.1 [4]: Let (D,W) be a default theory, let X be a subset of the set D of defaults and let S be a set of formulas.

1. We define $X_S = \left\{ \frac{\alpha}{\gamma} | \frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma} \in X,\ S \cup \{\neg\beta_i\} \text{ is inconsistent, } 1 \leqslant i \leqslant n \right\}$.

2. A set X of defaults is *active* with respect to W and S if it satisfies the conditions:

(i) Justif(X)=$\emptyset$ or Justif(X)$\cup$ S is consistent;

(ii) Prereq(X)$\subseteq Th^{X_S}$(W),

where Th^{X_S}(W) is the deductive closure of W using classical inference rules and the monotonic rules fromX_S.

We denote by A(D,W,S) the set of all subsets of the defaults in D which are active with respect to W and S. $\emptyset \subseteq$ A(D,W,S). MA(D,W,S) is defined as the set of all maximal elements in A(D,W,S).

The set E of formulas is a *rational extension* for the theory (D,W) if E=Th^{X_E}(W), where X$\in$ MA(D,W,E).

We observe that X is the *set of generating defaults* in this original definition of rational extensions.

The proof of this theorem consists in showing the following:

- Condition (i) from definition 2.1 and condition (i) from theorem 2.2 are equivalent, with the meaning: the reasoning context is consistent.
- Condition (ii) from definition 2.1 is equivalent with the condition of groundness for the set of generating defaults.
- Condition (ii) from theorem 2.2 is equivalent with the necessity to be maximal-active (from definition 2.1) for the set of generating defaults.

The proofs of these equivalencies are immediate.

The set D' from the theorems above is the *set of generating defaults* for the extension (E,C). Thus, both types of extensions are deductive closures of the set W (explicit content) and the consequents of D'.

The relationships between constrained and rational extensions are as follows:

- the set of rational extensions coincide with the set of constrained extensions for the class of seminormal theories (all defaults have the form $d = \frac{\alpha:\beta\wedge\gamma}{\gamma}$);
- every rational extension is a constrained extension of the same theory.

3. A theorem prover for propositional logic, based on a modified semantic tableaux method

The aim of the proposed propositional theorem prover is to verify the consistency/inconsistency of a propositional formula/set of formulas and to provide a model in the case of consistency. We will use at implementation level the symbols for the logical operations: $\sim$ ($\neg$), &($\wedge$), |($\vee$), > ($\rightarrow$), $-$($\leftrightarrow$).

This theorem prover is based on a modified version of the semantic tableaux method. We will use the same representation for a tableau, like the function TP [Schw90], as a set of sets of literals, but the construction of the tableau is different.

The semantic tableau $\cup_{i=1}^{n}\{\cup_{k=1}^{n_i}\{a_{ik}\}\}$ has n branches and corresponds to the disjunction of its branches. The i-branch of the tableau is a set of literals: $\cup_{k=1}^{n_i}\{a_{ik}\}$and represents the conjunction of its literals. The tableau corresponds to a formula with the disjunctive normal form $\vee_{i=1}^{n}\wedge_{k=1}^{n_i} a_{ik}$. If a branch contains a literal and its negation, we say that the *branch* is *closed,* otherwise the *branch* is *open.* If all the branches of a tableau are closed, the *tableau* is *closed*, otherwise the *tableau* is *open.*

All the open subtableaux of a semantic tableau T are called the *openings* of T.

The new idea is to construct the semantic tableau of a formula from its postfix form. Traversing the postfix form from left to right, using a stack mechanism to memorize partial semantic tableaux (corresponding to the subformulas of the formula), and applying operations to the tableaux, the construction of the semantic tableau is very simple and efficient.

Definition 3.1: Let denote by Tsem(F) the semantic tableau attached to formula F. We compute Tsem(F) as follows:

Tsem(a) = {{a}}, where 'a' is a propositional literal;

Tsem(∼F) = ∼Tsem(F), '∼' is negation

Tsem(F & G) = Tsem(F) & Tsem(G), '&' is conjunction

Tsem(F | G) = Tsem(F) | Tsem(G), '|' is disjunction

Tsem(F > G) = Tsem(F) > Tsem(G), '>' is logical implication

Tsem(F – G) = Tsem(F) - Tsem(G), '-' is logical equivalence

Definition 3.1 can be extended for computing the semantic tableau of a set of formulas as follows: Tsem({F1,F2,...,Fn}) = Tsem(F1)*Tsem(F2)*...*Tsem(Fn), where T1*T2 =T1&T2.

Definition 3.2: Let T1 = $\cup_{i=1}^{n}\{\cup_{k=1}^{n_i}\{a_{ik}\}\}$and T2 = $\cup_{j=1}^{m}\{\cup_{k=1}^{m_j}\{b_{jk}\}\}$ be two semantic tableaux. We define the operations ∼, &, |, >, - for semantic tableaux as follows:

∼T1={$\{\sim x_1, ..., \sim x_n\} \mid x_i \in \cup_{k=1}^{n_i}\{a_{ik}\}, i = 1, ..., n\}$

T1 | T2 ={$\cup_{k=1}^{n_i}\{a_{ik}\}$ | *i=1,...,n*} ∪ {$\cup_{k=1}^{m_i}\{b_{ik}\}$| *i=1,...,m*}

T1 & T2={$\cup_{k=1}^{n_i}\{a_{ik}\}$ ∪ $\cup_{k=1}^{m_j}\{b_{jk}\}$ | *i=1,2,...,n, j=1,...,m*}

T1 > T2 = ∼T1 | T2 and T1 – T2 = (T1 > T2) & (T2 > T1)

Example 3.1: Formula F=∼(a&b)|c&∼d has the postfix form ab&∼cd∼&|. Its semantic tableau, Tsem(F), is calculated step by step traversing the postfix form from left to right:

symbol	*partial semantic tableaux*	*stack*
'a':	T1=Tsem(a)={{a}},	st_tab=(T1)
'b':	T2=Tsem(b)={{b}},	st_tab=(T2,T1)
'&':	T3=T1 & T2={{a}}&{{b}}={{a, b}},	st_tab=(T3)
'∼':	T4=∼T3=∼{a&b}}={{∼a },{∼b}},	st_tab=(T4)
'c':	T5=Tsem(c)={{c}},	st_tab=(T5,T4)
'd':	T6=Tsem(d)={{d}},	st_tab=(T6,T5,T4)
'∼':	T7=∼T6=∼{{d}}={{∼d}},	st_tab=(T7,T5,T4)
'&':	T8=T5 & T7={{c}}&{{∼d}}={{c,∼d}},	st_tab=(T8,T4)
'|':	T9=T4 |T8={{∼a},{∼b}} | {{c,∼d}}= = {{∼a},{∼b},{c,∼d}},	st_tab=(T9)

Tsem(F)=T9={{∼a},{∼b},{c,∼d}}

The semantic tableaux method is a refutation method:

- formula F is valid (tautology) $\iff$ formula $\sim F$ is inconsistent $\iff$ Tsem($\sim F$) is a closed tableau;
- formula F is consistent $\iff$ Tsem(F) is an open tableau;
- formula G is deductible from the set $\{F_1,\ldots,F_n\} \iff \{F_1,\ldots,F_n,\sim G\}$ is inconsistent $\iff$ Tsem(F_1)*...* Tsem(F_n) * Tsem($\sim G$) is a closed tableau.

The main data structures used to implement the concepts: formula, set of formulas, semantic tableau, branch of a tableau are: *stiva, lista, formula, mult_formula, ramura* and *tabela*.

4. Implementation of *DARR* – a theorem prover for propositional constrained and rational default logic

We will consider in this paper only the case of propositional language as the underlying language for the default theories.

For easy access to the connection between the literals and the default where they belong, in the construction of a tableau all the literals are indexed as follows:

- the superior index is: **f** (literal from W) or **j** (literal from justifications) or **c** (literal from consequents).
- the inferior index is the number of the default where it belongs or is 0 if the literal belongs to the set of facts.

Adding indices to the literals from a semantic tableau we obtain an *indexed semantic tableau*, and we denote it by Tsem_ind.

The basic idea in computing constrained/rational extensions is to consider a maximal set X of formulas, $X = W \cup \text{Concl}(D) \cup \text{Justif}(D)$, that characterize the reasoning process (the facts, the consequents of the defaults and the justifications of the defaults) and then to supress the literals from defaults responsible for contradictions.

The candidates for the sets of generating defaults for extensions correspond to the openings in W of the indexed semantic tableau Tsem_ind(X).

All variants of default logic have in common the following property: the sets of generating defaults for extensions are grounded in the set of facts.

Definition 4.1: Let W be a set of formulas and let D be a set of closed defaults. We define the sequence of sets $(R_i)_{i\geqslant 0}$ as follows: $R_0 = \emptyset$ and

$$R_{i+1} = R_i \cup \left\{ d = \frac{\alpha : \beta}{\gamma} | d \in \text{D and } \text{W} \cup \text{Concl}(R_i)| = \alpha \right\}, i \geqslant 0.$$

The set D is *grounded in* W, if and only if $\text{D} = \bigcup_{i=0}^{\infty} R_i$.

D_baza $= \bigcup_{i=0}^{\infty} R_i$ is the maximal subset of D, grounded in W and can be calculated using algorithm *Submult_max_baza*(W,D,D_baza), which implements the above definition.

Using theorems 2.1 and 2.2 we can develop the following algorithm for computing all constrained and rational extensions of the default theory (D,W).

Algorithm 4.1:

Calcul_ext_restrictii_rationale(D,W)

begin

We construct the semantic indexed tableau:

T= Tsem_ind(W) *Tsem_ind(Justif(D))* Tsem_ind(Concl(D)).

We compute all the subsets $S_1, ..., S_n$ of D, such that the semantic tableaux:

Tsem_ind(W)* Tsem_ind(Concl(S_i))*Tsem_ind(Justif(S_i)), *i=1,. . . , n*

are open.

We eliminate from $S_1, ..., S_n$ the sets that are not maximal and we obtain the sets $R_1, ..., R_{n'}$ of defaults.

for i=1,. . . ,n' do

Submult_max_baza(W,R_i,R'_i)

endfor

print "(Th(W∪Concl(R'_i)),Th(W∪Concl(R'_i)∪Justif($'R_i$))) *i=1,. . . ,n*' are all constrained extensions, R'_i are the set of generating defaults"

if the theory (D,W) is semi-normal

then print "(Th(W∪Concl(R'_i)),Th(W∪Concl(R'_i)∪Justif(R'_i))) *i=1,. . . ,n*' are all rational extensions, R'_i are the set of generating defaults"

else

for i=1,. . . ,n' do

// we verify if R'_i is maximal active with respect to W and Th(W∪Concl(R'_i))

ind=0

while (ind==0 and not all$d \in D \backslash R'_i$ are chosen) do

We chose a new $d \in D \backslash R'_i$

if (Tsem(W) *Tsem(Concl(R'_i))*Tsem(Justif($R'_i \cup \{d\}$) is open

and Tsem(W) *Tsem(Concl(R'_i))*Tsem({∼Precond(d)) is closed
then ind=1
endif
endwhile
if ind==1
then print "R'_i cannot generate a rational extension "
else print "(Th(W∪Concl(R'_i)),Th(W∪Concl(D'_i)∪Justif(R'_i))) is a *rational extension* and R'_i is its set of generating defaults".
endif
endfor
endif
end

Accepting alternative possibilities for extending a default theory characterizes the *credulous reasoning*. The commonsense reasoning is the human model of reasoning, by making default assumptions for overcoming the lack of information. This type of reasoning belongs to the credulous perspective of the reasoning.

Skeptical reasoning is imposed in prediction problems because the nonmonotonic consequences cannot be later modified, which means that derived formulas does not depend on the alternative assumptions made during the reasoning process. It is considered irrational to have the possibility to chose one belief or another one if they are contradictory.

The specific of the problem will decide the appropriate perspective for the nonmonotonic reasoning used to solve the problem.

According to theorem 3.1 from [3] the skeptical nonmonotonic theorems of the theory (D,W) belong to the set: $Th^n_{D,\cap}$(W)=Th(W∪{$\vee_{i=1}^k \wedge_{j=1}^{n_i} c_j^i$}) where Concl($R_i$)={$c_1^i, c_2^i, ..., c_{n_i}^i$}, *i=1,. . . ,k*, and $R_1, ..., R_k$ are all the sets of generating defaults for the extensions of type *n=res (constrained) or n=rat (rational).*

If all extensions are calculated, the problem of membership to all extensions is reduced to a derivability problem in classical logic.

Algorithm 4.2:

Verif_consec_sceptica (f, D, W, mult_reg_gen)

begin
// mult_reg_gen={$R_1, ..., R_k$} from algorithm 4.1, Concl(R_i)={$c_1^i, c_2^i, ..., c_{n_i}^i$}, $i = 1, .., k$
if Tsem(W) *Tsem({$\vee_{i=1}^k \wedge_{j=1}^{n_i} c_j^i$})* Tsem({∼f}) is closed
then print "f is a skeptical nonmonotonic consequence of the theory (D,W)"
else print "f is not a skeptical nonmonotonic consequence of the theory (D,W)"
endif
end

The theorem prover for constrained and rational default logics is obtained by implementing the concepts: *indexed tableau, default, default theory* and the algorithms proposed above.

5. Conclusions

In this paper we have proposed a theorem for global characterization of rational extensions using the set of generating defaults and we have developed algorithms for solving the theorem proving problems specific for constrained and rational default logics.

The tight relationship between constrained and rational default logics was the reason to implement an automated theorem prover, called *DARR,* for both of these logic systems. The theorem prover proposed for propositional logic creates and manipulates in a very efficient and elegant way the semantic tableaux, using operators. A modified version of semantic tableaux method was adapted for computing constrained and rational extensions of a default theory.

References

[1] G.Antoniou, A.P.Courtney, J.Ernst, MA.Williams: *A System for Computing Constrained Default logic Extensions.* Logics in Artificial Intelligence, JELIA'96, Lecture Notes in Artificial Intelligence, 1126, 1996, pp. 237–250.

[2] P.Cholewinski, W.Marek si M.Truszczynski: *Default reasoning system DeReS.* Proceedings of KR-96, Morgan Kaufmann, 1996, pp. 518–528.

[3] M.Lupea: *Nonmonotonic inference operations for default logics.* ECIT 2002, Iasi, Romania, Symposium on Knowledge-based Systems and Expert Systems, pp. 1–12.

[4] A.Mikitiuk, M.Truszczynski: *Rational default logic and disjunctive logic programming*, în A. Nerode, L.Pereira, Logic programming and non-monotonic reasoning, MIT Press, 1993, pp. 283–299.

[5] P.Nicolas, F.Saubion, I.Stephan: *Genetic Algorithms for Extension Search in Default Logic*, 8-th International Workshop on Non-Monotonic Reasoning (NMR2000).

[6] R.Reiter: *A logic for default reasoning.* Journal of Artificial Intelligence, 13, 1980, pp. 81–132.

[7] T.H.Schaub: *Considerations on default logics.* Ph.D. Thesis, Technischen Hochschule Darmstadt, Germany, 1992.

[8] T.H.Schaub: *The Automation of Reasoning with Incomplete Information.* Springer-Verlag Berlin, 1997.

[9] T.H.Schaub: *XRay system: An implementation platform for local query-answering in default logics. Applications of Uncertainty Formalisms,* Lecture Notes in Computer Science, vol 1455, Springer Verlag, 1998, pp. 254–378.

[10] C.B.Schwind: *A tableaux-based theorem prover for a decidable subset of default logic.* 10-th International Conference on Automated Deduction. Lecture Notes in A.I. 449, 1990.

Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: lupea@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

THE CURVES ENCODING

VASILE PREJMEREAN AND SIMONA MOTOGNA

Abstract. In this paper we present an image description model that uses picture description language, and Bezier interpolation. We study 3-type images (represented by closed curves that conserve the critical points), giving modelling techniques, and also the coresponding algorithm. Our goal is to obtain a minimal description $\Pi_{\updownarrow}$-word of the smooth curves that aproximate the initial given curves.

1. Introduction

In this paper we present a method of encoding the 3-type images (according to the classification given in [7]) using $\Pi_{\updownarrow}$ -words to describe a set of critical points to which a Bezier interpolation is applied ([3,6,9]).

A 3-type image, described using a finite numbers of lines and curves, can be approximated through a 4-type image, described through a finite number of points in a cartesian rectangular system. The approximation consists of connecting the closest points to a curve $c \subset R^2$, points with integer coordinates, obtaining [5]:

$$M_{Pc}(c) = \{Apr(P) | P \in c\}, Apr : R^2 \to Z^2$$

These points can also be described by $\Pi_{\updownarrow}$-words.

Decoding means the operation of obtaining the curve closed to the initial one, and will be performed through interpolation, namely Bezier interpolation ([3,6,9]), of the points described by the $\Pi_{\updownarrow}$-words.

Bezier interpolation has been chosen because the resulting curves will be smooth, based on the two basic properties of these curves: they cross the initial and final points and are tangent to the initial and final segments determined by the first, respectively the last two points [7].

The curve we want to encode (and approximate) is divided in several curves, each of them initially described by a string of critical points, of length at most 7, according to the lemma 2.2 from [8]. In order to mark the breaking points of the curve, we will extend the description language Π^* (where the alphabet

2000 *Mathematics Subject Classification.* 68Q45, 68T10.

1998 *CR Categories and Descriptors.* I.5.2 [**Computing Methodologies**]: Pattern Recognition – *Design Methodology*.

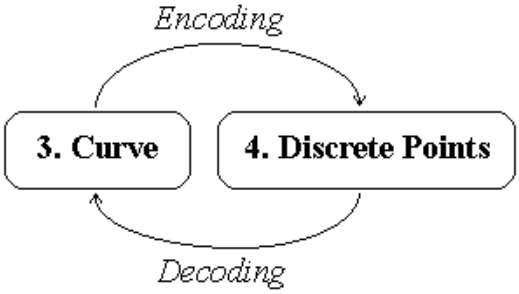

FIGURE 1. The transformations for the two image types

$\Pi = \{r, u, l, d\}$ stands for the commands *right, up, left, down*) to the language $\Pi_{|}^{*}$, where $\Pi_{|} = \Pi \cup \{|\}$, the command "|" representing the cut of the curve. These strings of length bigger than 7 will be reduced, eliminating some of them such that the curve resulted from interpolating the remaining points will preserve the critical points, thus approximating in an accurate way the given curve. The algorithm 2.1 tries to reduce the number of points such that it won't exceed 6. Since after elimination some points may no longer be neighbours, the description cannot be done through $\Pi_{|}$, and the language $\Pi_{\updownarrow} = \Pi_{|} \cup \{\uparrow, \downarrow\}$ will be needed, allowing points selection from a path traversed on the 4 directions.

2. The Set of Critical Points and Bezier Aproximation

From now on we will focus on 3-type images formed from closed curves, for example a system of level curves on a map as in Figure 2. The encoding of this image will be obtained by processing each curve; we will follow the process for the exterior curve, and the others will be treated similarly.

We will apply a rectangular network to the curve under study, such that every point from the curve (pixel from the image) will have two coordinates (x, y).

The string containing the centers of the squares that are crossed by the given curve will form the string of critical points (marked by small circles in Figure 3). The string of critical poits may start from anywhere, but it is more convenient to choose a starting point S such that together with its neighbouring points, they will be colinear. The explanation lies in the fact that S will also be the final point and the curve will be smooth, since a bezier curve is tangent to the starting and ending segments.

If this desirable situation is not possible, namely there aren't any three neighbour colinear points, then we will use a smoother network, halving the distance between horizontal and vertical lines and choosing one of the following situations:

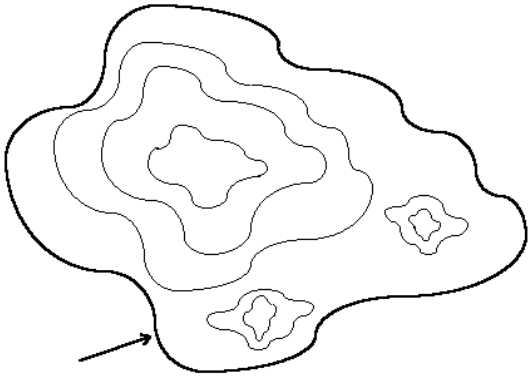

FIGURE 2. An example of a 3-type image

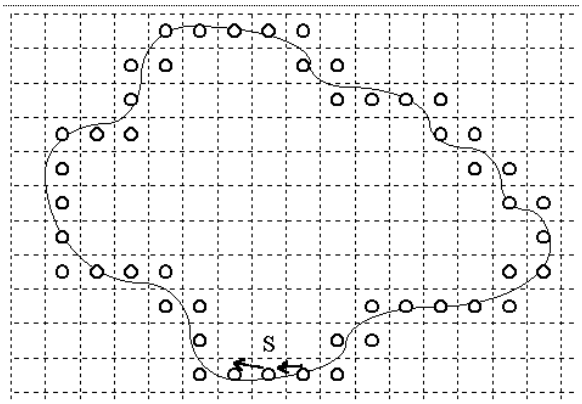

FIGURE 3. The string of critical points corresponding to a curve

a: re-compute the set of critical points and search a new starting point S;
b: insert a new point between any two neighbour points, at the middle distance between them; in this way any three succesive points will be colinear, and any point can be selected as start;

With this string of critical points we may approximate the given curve through a Bezier interpolation curve. The resulting curve (see Figure 4) will not be exactly what we've expected, since it does not approximate too well the initial curve. It does not preserve the critical points as shown in Figure 4.

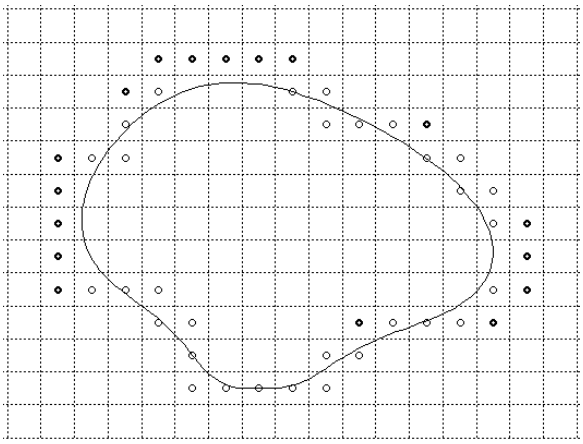

FIGURE 4. Initial curve and the one obtained through interpolation

Applying a new approximation we will get further to the initial curve C_0, although it will be desirable to obtain the same C_B, constant from now on.

Remark 2.1: One may notice that if from the set M_{P_C} of critical points we obtain the curve C_B applying a transformation T, and from the curve C_B we obtain the same set M_{P_C} of critical points applying a transformation T_1, then these two transformations represent each others inverses ($T_1 = T^{-1}$ and $T = T_1^{-1}$):

$$C_0 \to M_{P_C} \overset{\mathrm{T}}{\to} C_B \overset{\mathrm{T}^{-1}}{\to} M_{P_C} \overset{\mathrm{T}}{\to} C_B ...$$

The possibility of reducing the number of critical points is studied for each subcurve (the example from Figure 5 studies the curve between S_4 and S_1). In figure 6 you may notice that removing the 4 critical points denoted by "x" we obtain a Bezier approximation curve that satisfies the two proposed properties.

In order to obtain the desired approximation curve we will apply *Bezier* algorithm for each subsequence, and the resulting curve is obtained unifying the s curves (in Figure 5 we have 4 curves divided by the points $S_1, ..., S_4$ determined by these subsequences: $C_B(P) = Bezier(P_1^1, ..., P_{n1}^1) \cup Bezier(P_1^2, ..., P_{n2}^2) \cup ... \cup Bezier(P_1^s, ..., P_{ns}^s)$, where $M_{P_C}^k = \{P_1^k, ..., P_{nk}^k\}$ is the set of critical points obtained for the curve k applying algorithms similar to the ones we will describe below.

The result can be further improved if we can reduce even more the number of points, even if we choose other points, not just removing the initial ones. In the following, we will study this posiibility, trying to reduce the number of remaining points (the nine points from figure 6 can be reduced to the five points from figure

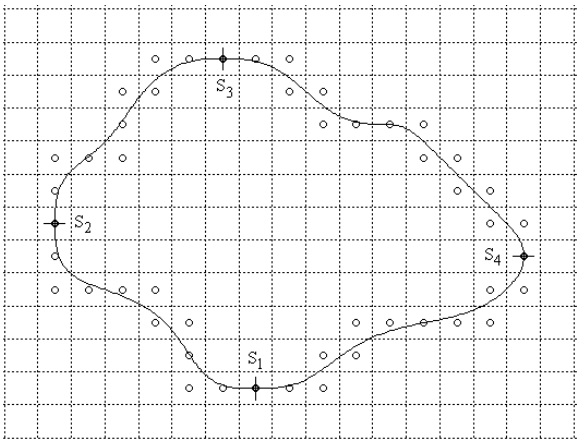

FIGURE 5. Dividing the curve

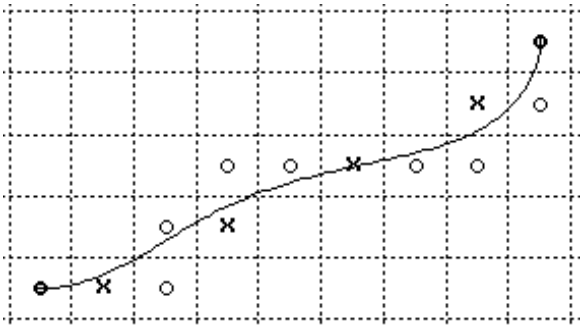

FIGURE 6. Reducing the number of critical points

7, denoted $P_1, P_2, ..., P_5$. On this specific example, the curve description can be obtained using only five interpolation points.

There are some remarks to be made analyzing figure 7: the initial point P_1 and the final point P_5 must be preserved, the second point P_2 and the penultimate point P_4 must be on the same direction with the old points (in order to satisfy the smoothness property). The point P_3 had been choosen such that the obtained curve preserves the critical points (meaning also that be obtain a valid approximation of the given curve).

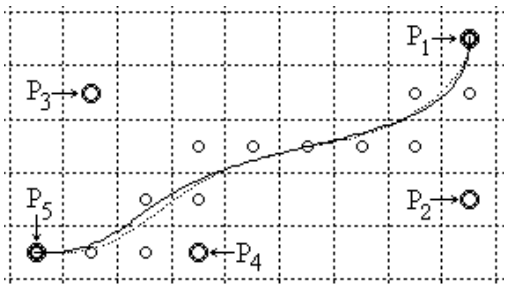

FIGURE 7. Approximation using 5 points

Algorithm 2.1

The following algorithm tries to find the points $P_1, ..., P_n$ that describe the Bezier approximation curve corresponding to the given critical points $Q_1, ..., Q_m$, starting from 2, 3,..., 6 points. If we don't succeed, the curve must be divided again (recommended to).

Begin
Input $Q_1, Q_2, ..., Q_m$;
if (Q_1, Q_m) *can describe the curve* **then** n:=2;
Output Q_1, Q_m
else
if *exists a point R such that* (Q_1, R, Q_m) *can describe the curve*
then n:=3;
Output Q_1, R, Q_m
else
if *exist 2 points R,S such that* (Q_1, R, S, Q_m) *can describe the curve*
then n:=4;
Output Q_1, R, S, Q_m
else
n:=5;
$P_1 := Q_1; P_n := Q_m$;
Find (P_2 *on direction* $Q_1 \to Q_2$) **and** (P_{n-1} *on direction* $Q_m \to Q_{m-1}$) **and**
Find (P_3 *in the domain* (P_1, P_n) *depending on* (P_2, P_{n-1}));
if $Ok(Q_1, Q_2, ..., Q_m, P_1, P_2, ..., P_n))$ **then Output** $P_1, ..., P_n$
else
n:=6;
$P_1 := Q_1; P_n := Q_m$;
Find (P_2 *on direction* $Q_1 \to Q_2$) **and** (P_{n-1} *on direction* $Q_m \to Q_{m-1}$) **and**
Find (P_3 *and* P_4 *in the domain* (P_1, P_n) *depending on* (P_2, P_{n-1}));
if $Ok(Q_1, Q_2, ..., Q_m, P_1, P_2, ..., P_n))$ **then Output** $P_1, ..., P_n$

else write('The curve must be divided')

End.

We present an analysis of the performed steps:

a) For $n = 2$ - it is very simple, because it is easy to verify if the segment Q_1Q_m is horizontal or vertical and traverse the critical point.

b) For $n = 3$ - we must verify if the triangle Q_1RQ_m is rectangular or not, and the critical points are preserved (there are two possibilities $R(x_1, y_m)$ or $R(x_m, y_1)$).

c) For $n = 4$ - is not dificult to find the points $R(= P_2)$ and $S(= P_3)$, because R must have the coordinates (x_1, y) or (x, y_1) and S must have the coordinates (x_m, y) or (x, y_m), where $x \in (x_1, x_m)$ and $y \in (y_1, y_m)$. There are the four possibilities presented in figure 8.

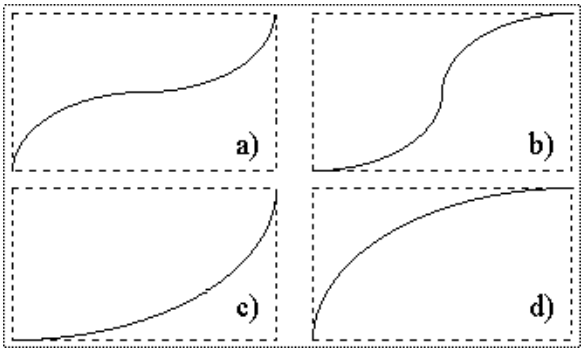

FIGURE 8. The 4 cases using 4 approximation points

d) For $n = 5$ - one can easy see that if we use only the four points P_1, P_2, P_4, P_5 to approximate the curve from figure 7 we obtain a curve like the one from figure 8 c). Then, we need a point (like P_3, from figure 7) to "drag" the curve. Finally, the points P1, P2, P3, P4, P5 approximate correctly the given curve. If we consider the case c) from figure 8, finding $P_2(x_1, y)$ on direction $Q_1 \rightarrow Q_2$ means to find a value $y < y_1$, because Q_1 have the coordinates (x_1, y_1) and Q_2 have the coordinates $(x_2 = x_1, y_2)$ and analogues, finding $P_{n-1}(x, y_m)$ on direction $Q_m \rightarrow Q_{m-1}$ means to find a value $x > x_m$, because Q_m have the coordinates (x_m, y_m) and Q_{m-1} have the coordinates $(x_{m-1}, y_{m-1} = y_m)$. The search of the desired points P_1 and P_{n-1} is performed alternatively increasing (or decreasing) the coordinate y starting from the initial value $y_1 \pm 1$, and the coordinate x starting from $x_m \pm 1$ until the desired points are obtained or an imposed maximum value is overflow (i.e., x_1, respectively y_m). Finding $P_3(x, y)$ in the domain $(P_1(x_1, y_1), P_n(x_m, y_m))$ means to find a value x such

that $x_m < x < x_1$ and a value y such that $y_m < y < y_1$ (for our case c) from figure 8).

e) For $n = 6$ - the points P_1, P_2, P_5, P_6 will be constructed like P_1, P_2, P_4, P_5 (see case n=5) and after that we need two points $P_3(x, y)$ and $P_4(x', y')$ in the domain $(P_1(x_1, y_1), P_n(x_m, y_m))$ with the same properties like above (like P_3 for n=5).

f) For $n > 6$ - it is more efficient to cut the curve and to apply this procedure on each of the parts.

The number of curves in which the initial curve is decomposed can be reduced in the following way (as in Figure 9):

- eliminating some cutting points: existing colinear points that would solve the problem correctly;
- searching certain critical points even outside the domain (less points).

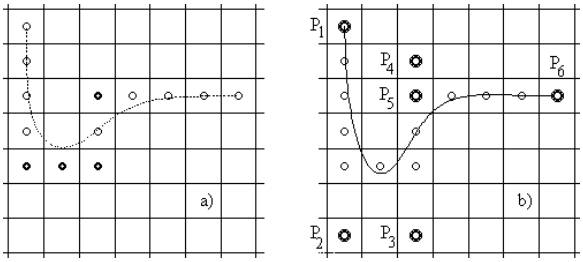

FIGURE 9. The curves obtained with 6 points

As shown in Figure 9a, if during interpolation we consider all points, then the obtain curve does not preserve the critical points (the three colinear points from the bottom), but the six points from Figure 9b preserve the curve.

The two interpolation pointsfrom the bottom are needed in order to "drag" the curve through the uncovered critical points. The point P_3 is inside the domain and is needed in order to cover the critical point that remained uncoverd by the curve from figure 9a.

Searching the points P_2, P_3, P_4, P_5 is quite a difficult problem, if we take into account the eficiency of the algorithm, because the points P_2 and P_5 must be searched in the form: $P_2(x_1, y_1 + Dy), P_5(x_m - Dx, y_1)$ and the points $P_3(x_3, y_3)$ and $P_4(x_4, y_4)$ must be searched in the domain:

$$(min(X), max(X)) \times (min(Y), max(Y)),$$

$$\text{where } X = \{x_1, x_2, x_5, x_6\} \text{ and } Y = \{y_1, y_2, y_5, y_6\}$$

Then, the searching algorithm has the following structure:

Algorithm 2.2

(1) **for** Dx:= 1 **to** $LimX$ **do**
(2) **for** Dy:=1 **to** $LimY$ **do**
(3) **for** x_3 := *min(X)* **to** *max(X)* **do**
(4) **for** y_3 := *min(Y)* **to** *max(Y)* **do**
(5) **for** x_4 := *min(X)* **to** *max(X)* **do**
(6) **for** y_4 := *min(Y)* **to** *max(Y)* **do**
(7) **if** *Ok(P,Q)* **then output** P

where $\mathbf{P} = (P_1, P_2, ..., P_6), \mathbf{Q} = (Q_1, Q_2, ...Q_m)$

This algorithm determines all solutions, but is inefficient due to the six loops. The optimizing of this algorithm can take into consideration that we may be satisfied with only some of the solutions or even with one solution in exachang to a more efficient search.

A first direction for optimization is to unify the loops from lines 3 and 4, respectively from lines 5 and 6. In this way, the search of the points P_3, P_4 is restricted on the directions in which the critical points haven't been covered (in our example, to the rigth, respectively upwards). Now, the lines 3 to 6 will be replaced with:

For Dr:=1 To Lr Do
For Du:=1 To Lu Do

In this situation, in our example, the points P_3 and P_4 will be search in the form $P_3(x_2 + Dr, y_1)$, respectively $P_4(x_5, y_5 - Du)$. Of course, applying such a strategy does not assure that all solutions will be obtained. This example will generate two solutions: the one in figure 9b and from figure 10a. The other three solutions from figure 10 (b,c,d) are not on that direction and we have to extend the search to a neighbourhood (inside the domain) of the points P_3 and P_4, if we haven't obtained any solution.

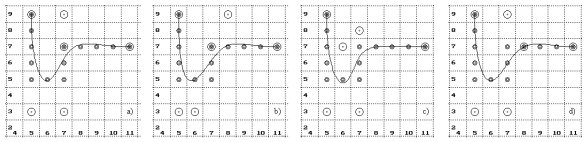

FIGURE 10. Other curves obtained with 6 points

In order to appreciate how close a Bezier curve $Bezier(P)$ is to the set of critical points Q, we define the distance $\delta(P, Q)$ as:

$$\delta(P, Q) = Nr.Minus + Nr.Plus$$

where:

- $Nr.Minus$ represents the number of critical points uncovered by the approximation curve $Bezier(P)$

- $Nr.Plus$ represents the number of extra points covered by the approximation curve ($\notin Q$)

Then, the distance $\delta(P, Q)$ is the cardinal of the simetric difference (Δ) between the set of *representative* points for the Bezier curve (C_B) corresponding to the determined points P) and the set of given critical points ($M_{P_C} = Q$):

$$\delta(P, Q) = |\{Apr(B)|B \in Bezier(P)\} \ \Delta \ Q|$$

For example, if we consider the approximation from figure 11, obtained for $Dx = 4(P_5(7, 7)), Dy = 5(P_2(5, 4)), Dr = 2(P_3(7, 4))$ and $Du = 1(P_4(7, 8))$, the distance is $\delta(P, Q) = 2 + 1 = 3$, since the points (5,5) and (7,5) are not covered, and the point (6,6) should not be included.

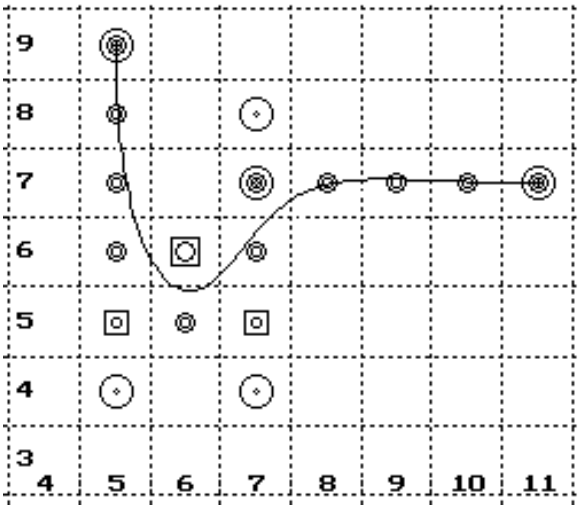

FIGURE 11. An example of approximation

If we study the minimal values obtained for the pairs (Dy, Dx) from table 2.1 we can notice that for $(Dy, Dx) \in \{(6, 3), (6, 4), (6, 5)\}$ there exist solutions, because the minimal values for the distance δ are zero. These minimal values are obtained choosing P_3 in the neighbourhood of P_2, respectively choosing P_4 in the neighbourhood of P_5. Another remark is that these values are grouped in a zone, to which if we get farther, then the values increase.

This remark gives the possibility to limit the values $LimX$ and $LimY$ from the loops (1) and (2), since the more we get farther from the zero positions, the more the values of distances increase.

Even more, if we re not interesting in obtaining all solutions and one soltuion is enough (as the approximation problem was initially stated), for example the

Min(δ)	*Dx:* **1**	**2**	**3**	**4**	**5**	**6**
Dy: **1**	12	10	10	8	8	8
2	8	8	8	6	6	6
3	6	6	6	6	6	6
4	4	4	4	4	5	5
5	6	4	3	2	3	4
6	2	2	0	0	0	4
7	8	3	2	1	1	3

TABLE 1. The distances obtained for the given curve

position (6,3), then the search should not parse the entire matrix, but the domain $(1,1) \times (6,3)$, so $LimX = 3$ and $LimY = 6$. The search can be performed succesively adding square matrixes, in the order given in table 2.2. One may notice that in the fifth column (or even earlier) the values of δ increases upwards, so it is possible to quit searching the solution in that zone and move to the next line.

Searching order	*Dx:* **1**	**2**	**3**	**4**	**5**	**6**
Dy: **1**	1	4	9	16	25	-
2	2	3	8	15	24	-
3	5	6	7	14	23	-
4	10	11	12	13	22	-
5	17	18	19	20	21	-
6	26	27	**28**	...	...	-
7	-	-	-	-	-	-

TABLE 2. The searching order - first version

Of course, a *Branch and Bound* algorithm will be more suitable, especially since we have already defined a distance δ to the solution (if $\delta(P,Q) =)$ then P is a solution). We will characterize a state through the components Dx, Dy, d_1 and d_2. The ditance d_1 represents the computation step, and $d_2 = \delta(Dx, Dy)$. The initial state will be $(1,1,1,\delta(1,1))$, and the final state will be characterized by $\delta = 0(d_2 = 0)$.

From a current state (choosen from the set of active states, where the minimum of the sum $d_1 + d_2$ is obtained) we will generate two active states $(Dx+1, Dy, d_1 + 1, \delta(Dx+1, Dy))$ and $(Dx, Dy+1, d_1+1, \delta(Dx, Dy+1))$, and this current state will become pasive. The steps to be performed can be studied in table 2.3, and it is also easy to remark that the solution will be obtained in 8 steps. If the set of active states become empty, such that no new selection of a current state can

be performed, then the problem has no solution. In order to end the algorithm in such a case we must specify a limit $LimX$ for Dx and a limit $LimY$ for Dy.

Searching order	*Dx:* **1**	**2**	**3**	**4**	**5**	**6**
Dy: **1**	**1**+12	2+10	10	8	8	8
2	**2**+8	3+8	8	6	6	6
3	**3**+6	4+6	6	6	6	6
4	**4**+4	**5**+4	6+4	4	5	5
5	5+6	**6**+4	7+3	2	3	4
6	2	**7**+2	**8+0**	0	0	4
7	8	8+3	2	1	1	3

TABLE 3. The searching order - *Branch and Bound* approach

A last remark is that in the case in which there are more than one active state with the same minimal value for $d_1 + d_2$, we will choose the one for which d_2 is minimal. As shown in table 2.3, at the step 7 we have "preferred" the state (2,5,6,4) (or (3,4,6,4)) although the state (2,3,4,6) has the same value for $d_1+d_2 = 10$, since is closer to the solution (d_2 approximates the distance between a state and the final state). Even if we would have choosen the other state, the algorithm gives the desired solution, but an extra step would have been performed, and then come back and choose the "preferred" state.

3. Using Π-words to describe pictures of 3-type

In this section, we present a model of approximating a curve using a picture description language. At the beginning, we will use the alphabet $\Pi = \{r, u, l, d\}$ to describe the movement of the pen on the four directions, then we will extend the commands alphabet to $\Pi_| = \Pi \cup \{|\}$ (where | denotes an interuption of the sequence ofcritical points) in order to define the cuts of a curve; eventually, this alphabet will be enhanced with two more commands *pen-up* and *pen-down* $\Pi_{\downarrow} = \Pi_| \cup \{\uparrow, \downarrow\}$, in order to avoid (eliminate) certain points, reducing the number of critical points that will be interpolated obtaining an approximation curve.

If we consider the example from figure 5, then the Π-word $w \in \Pi^*$ is:

$$\boldsymbol{w = lluulullluu\ uurruururr\ rrdrdrrrdrdrdrd\ dldlllldldll}$$

The insertion positions, corresponding to the possible interuption points (points between two colinear points), can be anywhere between two identical commands(characters). If this is not possible or is not convenient, then the word w can be easily modified doubleing each character. In this manner, the curve can be described starting from any point and can be interrupted at any moment, since any three consecutive points are colinear. The description word becomes:

$$w' = l^4u^2(u^2l^2)^2l^4u^8r^4u^2(u^2r^2)^2r^6(d^2r^2)^2r^4(d^2r^2)^3d^4l^2d^2l^8(d^2l^2)^2l^2.$$

Since our example doesn't need any network or command word doubleing, we will work from now on with the word w (not w').

Since the command sequence is circular [1], we can execute it from any position, and then come back to the beginnign of the string and execute the remaining commands. The start command will be a character that has a predecessor (left neighbour) equal to it (the points must be colinear).

Even more, for a selected string, marked with interruption commands, denoted | and that will cut the string into substrings, corresponding to the curves that forms the given closed curve; for example, the $\Pi_{|}$-word that describe the curve from figure 5 is

$\boldsymbol{w = llulullluu|uurruurur|rrdrdrrxrdrdrdrd|dldlllldldll}$

we can introduce avoiding commands (eliminating critical points), that will preserve the quality of the curve. (the smoothness and the critical points).

We can use the alphabet $\Pi_{\updownarrow}$ that allows the construction of $\Pi_{\updownarrow}$-words (containing two more symbols $\uparrow = pen - up$ and $\downarrow = pen - down$) : $\Pi_{\updownarrow} = \{r, u, l, d, |, \uparrow, \downarrow\}$. These words will describe a sequence of points that determines a curve by Bezier interpolation. This means that a set of $\Pi_{\updownarrow}$-words describes curves that compose a 3-type image. The $\Pi_{\updownarrow}$-word for the curve from figure 6 is $d \uparrow l \downarrow dl \uparrow l \downarrow ll \uparrow d \downarrow ld \uparrow l \downarrow l$, and for the curve from figure 7 the Π-word is $\uparrow d^2 \downarrow d \uparrow l^7 u \downarrow u \uparrow d^3 r \downarrow r \uparrow l^2 \downarrow l$. An even simpler description convention can be used, if the points are rare, as in our example, if we remove the sequence that denote a point of the form $\downarrow \tau \uparrow$ (where $\tau \in \Pi$) and replace it with a single command character (for example $\downarrow$), that attach the current point to the sequence of interpolation points. This means that, implicitely, the movement of the pen is done "without drawing", and when a $\downarrow$ command is met, the current point will be stored. We could also consider that these characters are present at the beginning and at the end of the description word. In the example from figure 7, the description word may be $\downarrow d^3 \downarrow l^7 u^2 \downarrow d^3 r^2 \downarrow l^2 \downarrow l^3 \downarrow$ or if we give up the first and last character, we obtain a reduced description: $w = d^3 \downarrow l^7 u^2 \downarrow d^3 r^2 \downarrow l^2 \downarrow l^3$.

Reducing the number of critical points also implies the preserving of the initial points, so the fixed point problem for a new approximation (curves convergence) is solved.

REFERENCES

1. F.J. Brandemburg, M.P. Chytil, On Picture Languages : Cycles and Syntax - Directed Transformations, Technische Berichte der Fakultat fur Mathematik und Informatik Universitat Passau, MIP-9020, 1990.
2. J. Dassow, F. Hinz, Decision problems and regular chain code picture languages, Discrete Applied Mathematics, no.45, 1993, pp. 29-49.
3. J.D. Foley, A.V. Dam, Fundamentals of Interactive Computer Graphics, Addison Wesley, London, 1982.

4. H.A. Maurer, G. Rozenberg, E. Welzl, Using String Languages to Describe Picture Languages, Information and Control, Vol.54, Nr.3, 1982, pp.115-185.
5. S.Motogna, V.Cioban, V.Prejmerean, Picture Approximation, Studia Univ. Babes-Bolyai, Vol.XLIII, Nr.2, 1998, pp.43-55.
6. T. Pavlidis, Algorithms for Graphics and Image Processing, Springer-Verlag, Berlin-Heidelberg, 1982.
7. A. Rosenfeld, Picture Processing by Computer, Academic Press, London, 1969.
8. I.H. Sudborough, E. Welzl, Complexity and Decidability for Chain Code Picture Languages, Universitat Graz, F125, 1983.
9. A. Watt, 3D Computer Graphics, Addison-Wesley, Great Britain, 1993.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania

E-mail address: per|motogna@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

UML MODEL CHECKING

DAN CHIOREAN, ADRIAN CARCU, MIHAI PASCA, CRISTIAN BOTIZA, HORIA CHIOREAN, AND SORIN MOLDOVAN

Abstract. Correctness against the UML definition has to be a prerequisite for every UML model. In terms of programming languages this requirement is stated: "the precondition for every application is to be syntactically and semantically correct against the language specification". The objective of this paper is to go over the state of the art in this domain highlighting some drawbacks in the UML 1.4 AO and WFR[1]. The XMI adoption as a standard format for UML models transfer opened the way to verifying the level at which different UML tools comply with the UML semantics. Taking into account that existing OCL tools do not implement all the functionalities required for efficient UML model checking, we have designed and implemented an OCL evaluator[2]. The possibility to check every UML Model stored in XMI format, a repository fully compliant with UML 1.4, including all the AO, the possibility to evaluate the WFR, MR[3] and BCR[4], are among the main features of our tool.

Key words: UML 1.4, OCL, UML model checking, CASE Tools, AO, WFR, OCL evaluator

1. The UML model correctness

UML model correctness is certainly a very important aspect unfortunately ignored by many specialists in the modeling domain. How else could we possibly explain a series of errors found in different UML models, which the user is not warned about after the check?

By model correctness, we understand the correctness of the model against the modeling language. For UML this means satisfying the WFR. Further, after the WFR have been passed, the BCR have to be syntactically and semantically correct. Asking that different kinds of applications comply with a set of Methodological Rules, may extend the concept of "correctness". An aspect related with model checking is the moment at which the check is performed. In this paper, we are referring to key moments in the application's life cycle; for example the moment

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

when the design model is automatically turned into code, the moment right before the end of modeling, the moment before exporting the model to another tool etc. The importance of model validation in the above-mentioned key moments is unanimously recognized. For example, in his paper "Consistency Checking" [Moore2000], Michael Moors states: "... during model editing, the model will frequently be syntactically incorrect, and the tool needs to be able to allow for syntactical incorrectness in this mode." The Amigos also call this "Inconsistent models for work in progress," because "the Final model must satisfy various validity constraints to be meaningful."

To understand better the value of UML models correctness, we suggest you reflect over the syntactic and semantic correctness of an application against the implementation language. In this later case, the existence of errors does not allow you to build the application. The process will be interrupted at compile or link time. Using the existent CASE Tools, the errors identified in the analysis or design models, do not interrupt the development process immediately. Consequently, errors are transferred to the next model produced in the software lifecycle. This implies a bigger amount of time and money spent on application development.

2. State of the art

To test UML model correctness, at least all the WFR must be evaluated. Naturally the "precondition" for the above statement is: The WFR have to be correct and complete (in other words, they have to cover at least all the important semantic features of the UML model elements).

The usage of formalisms such as script languages or formal languages (different from OCL) has some drawbacks. In order to be rigorous, we have to demonstrate for each WFR, the equivalence between its specification in OCL and its specification in the used formalism. (Supposing that WFR are correctly specified in OCL). Another problem, even more embarrassing, can appear when the checks are done using a UML CASE tool. This is because even in the best case (when the CASE Tool repository fully implements the UML metamodel), the tools don't allow the user to create some model elements or to declare certain relationships among the existent model elements. The most used CASE Tools – Rational Rose, Together, Poseidon, etc. haven't yet implemented: the Inheritance Relationship among packages, the Permission Relationship between packages (including standard stereotypes for this relationship), the Collaboration ModelElement, and other concepts defined in UML 1.4. Moreover, the Repository Interface for the abovementioned tools is pretty different from the interface formed by joining the `get` and `set` operations defined for the UML metamodel classes and the AO. (Our position is that a minimal UML repository interface should include at least the AO and the set and get operations)

Among the existent CASE Tools offering OCL support (`http://www.klasse.nl/ocl`) neither Argo (Poseidon) nor Use or ModelRUN don't provide user access

to the tool repository by means of AO. Consequently they do not support UML model checking in a straightforward manner.

In some UML papers, and particularly in [Richters 2000] there are mentioned different drawbacks of the UML AO and WFR. Most of them are syntactic and semantic errors. Unfortunately, there are also some conceptual errors. In the following, we will try to focus on this category of errors.

We noticed that the UML 1.4 "static semantics", expressed using OCL expressions, contains a lot of errors. Consequently, we will try to find the rationale of this situation and propose some solutions.

3. The LCI OCL Evaluator

Because the main objective of this paper is OCL, AO and WFR, we will not insist on our tool architecture. Below we roughly present how to use our checker.

As we can see in Figure 1, the main components of this tool are: the repository, the XML reader, the OCL/UML Type System, the syntactic analyzer, the semantic analyzer, the evaluator and the GUI.

First, the user has to load the model, the UML 1.4 metamodel (both expressed in XMI format) and the WFR or other constraints or operations specifications, expressed in a text with the ".ocl" extension. The succession of these three operations does not matter.

Before beginning the checking process, the user has to verify the OCL expressions syntactically and semantically. In case of semantic errors, the tool offers the possibility to do a partial evaluation. In this process, the type of expressions located before the error can be calculated.

The next step consists in identifying the model element(s) to be checked next against an OCL constraint. Finally, the last step consists in constraint evaluation. As we mentioned before (in case of semantic errors), the user has the possibility to evaluate the whole expression or parts of them.

For the moment we are, the user has just the possibility to modify the OCL expressions. To correct (change) the UML models, he has to use the UML CASE Tools and to save the modified models in XMI format, in order to do a new check. To evaluate dynamically the OCL constraints we intent to translate the OCL specifications in a programming language (Java, C++ etc.) and to generate automatically the code for the UML models. Concerning the methods code, this will be write by hand or generate automatically using the State Transition Diagrams or Object Diagrams.

Taking into account the aspects mentioned in the previous section, one of our main objectives was to support the user in checking UML models. In order to do this, the OCL constraints and specifications can be evaluated both at the metamodel and model level.

4. Errors in AO specifications

First of all we will analyze the operation pair contents, allContents, defined in the Namespace context.

As is very well mentioned in [Richters 2000] both operations have the same specification in English and in OCL. A first observation we made is that the specification below, can not be evaluated because the stop condition is not explicitly mentioned.

contents: Set(ModelElement)

contents = self.ownedElement->union(self.namespace.contents)

In order to evaluate the contents AO defined in the Namespace context, we propose:

contents = **if** self.namespace->isEmpty **then** self.ownedElement **else**

self.ownedElement->union(self.namespace.contents) **endif**

We found this specification clearer even in case of implementing manually in the UML 1.4 API repository. The second remark concerns the expressiveness of the operation's name, directly connected to the contents "specification" in English language. "The operation contents results in a Set containing all ModelElements contained by the Namespace".

For us, the above operation, return the Set of ModelElements visible (potential servers) in a Namespace if we do not take into account the dependency relationship among the Namespace and other server Namespaces.

Analyzing the UML AO, we notice that the above operation is redefined in the Classifier context and in the Package context, where the inherited elements are tacked into consideration. The specifications are identical in both cases. Taking into account that both Classifier and Package are descendents of GeneralizableElement, our opinion is that it would be better to define allContents operation only in GeneralizableElement. In this case, the conflict existent in Subsystem, do to a multiple inheritance of allContents operation both from Classifier and Package disappear.

The operation is equally redefined in Collaboration, in order to rejected the elements specialized in descendants. In this case, the potential conflict due to a multiple inheritance of allContents operation in the GeneralizableElement is solved due to the above mentioned redefinition.

Another example is provided by the specification of allFeatures AO. In this case, there is discordance between the "specification" (description) made in English language and those made in OCL. In [UML 1.4] is stated: "The operation allFeatures results in a Set containing all Features of the Classifier itself and all its inherited Features.", the OCL specification being:

allFeatures =

self.feature->union(self.parent.oclAsType(Classifier).allFeatures)

We can simply notice that in the OCL specification the ancestors' private features had not been eliminated. The correct specification can be (the reject operation can also tacked into consideration):

```
allFeatures = self.feature->union(self.parent.oclAsType(Classifier)
.allFeatures->select(f | f.visibility=#public or f.visibility=#protected))
```

The allFeatures is a very important specification because she is used in allOperations, allMethods and allAttributes AO and in different WFR.

```
allContents = self.contents->union(self.parent.allContents->select(e |
e.elementOwnership.visibility = # public or e.elementOwnership.visibility =
# protected))
```

5. Errors in WFR

In the following we will analyze the WFR using the above mentioned AO, beginning with the WFR[4] defined in the Association context.

The connected Classifiers of the AssociationEnds should be included in the Namespace of the Association, or be Classifiers with public visibility in other Namespaces to which the Namespace of the Association has "access" Permissions.

```
self.allConnections->forAll(r | self.namespace.allContents->includes
(r.participant)) or
self.allConnections->forAll(r | self.namespace.allContents->excludes
(r.participant) implies
self.namespace.clientDependency->exists(d | d.oclIsTypeOf(Permission) and
d.stereotype.name = 'access' and
d.supplier.oclAsType(Namespace).ownedElement->select(e |
e.elementOwnership.visibility = #public)->includes(r.participant) or
d.supplier.oclAsType(GeneralizableElement).
allParents.oclAsType(Namespace).ownedElement->select(e |
e.elementOwnership.visibility = #public)->includes(r.participant) or
d.supplier.oclAsType(Package).allImportedElements->select(e |
e.elementImport.visibility = #public)->includes(r.participant)))
```

Apart from the specification form, we notice that the classifiers inherited were taken into consideration twice. These because the association's Namespace has to be a Package and, as we mentioned in the previous section, in the Package, the allContents operation had been redefined in order to include the inherited elements.

```
allContents = self.contents->union(self.parent.allContents->select(e |
e.elementOwnership.visibility = #public or e.elementOwnership.visibility =
#protected))
```

More, the Permission relationships having the stereotype 'friend' and 'import' have not been taken into consideration. In this case, our proposal is to define in the Package context an allVisibleElements AO in order to return all the elements able to be used as servers in that Package. This AO will be useful to check all the

relationships defined in that Package and to calculate the potential servers for the features of Classifiers defined in this Package.

```
context Package::allVisibleElements():Set(ModelElement)
post:
let clDepSuplElem(d: Dependency): Set(ModelElement)= d.supplier->asSequence
->first.oclAsType(Package).ownedElement
let clDepStName(d: Dependency):String = d.stereotype->asSequence->first
.name in
allVisibleElements() = self.allContents->union(self.clientDependency->
select(oclIsKindOf(Permission))->iterate(cD ; acc:Set(ModelElement)=Set{} |
    if (clDepStName(cD)='import' or (clDepStName(cD)='access'))
        then acc->union(clDepSuplElem(cD)->select(e | e.oclIsTypeOf(Classifier)

            and e.elementOwnership.visibility=#public))
        else if (clDepStName(cD)='friend')
            then acc->union(clDepSuplElem(cD)->select(oclIsTypeOf(Classifier)))
            else acc->union(Set{})
        endif
    endif))
```

In this case, the Association WFR[4] will be:

```
context Association
inv WFR_4:
(self.namespace.allVisibleElements.oclAsType(Classifier)->asSet)
->includesAll(self.connection->iterate(ae ; acc:Set(Classifier)=Set{} |
acc->including(ae.participant)))
```

In order to support our proposal, we will analyze now, the BehavioralFeature WFR[2].

"[2] The type of the Parameters should be included in the Namespace of the Classifier."

```
self.parameter->forAll(p | self.owner.namespace.allContents->includes
(p.type))
```

It is very clear that in this case, the classifiers visible by means of relationships declared among the classifier's package and other packages were not be tacked into consideration. The above proposed allVisibleElements AO is proving to be useful in this context also.

Another aspect worth to be analyzed is the Class WFR[2].

"A Class can only contain Classes, Associations, Generalizations, UseCases, Constraints, Dependencies, Collaborations, DataTypes, and Interfaces as a Namespace."

```
context Class
inv WFR_2:
self.allContents->forAll->(c | c.oclIsKindOf(Class) or c.oclIsKindOf(Association)
or c.oclIsKindOf(Generalization) or
```

c.oclIsKindOf(UseCase) **or** c.oclIsKindOf(Constraint) **or**
c.oclIsKindOf(Dependency) **or** c.oclIsKindOf(Collaboration) **or**
c.oclIsKindOf(DataType) **or** c.oclIsKindOf(Interface))

Concerning this WFR there are at lest two observations to do. Firstly, supposing that this WFR really has a role, the invariant is incomplete because at least the Realisation relationships between an interface and a Classifier, the Derivation relationships and the AssociationClasses were forgotten.

Due to the above mentioned lacks (errors) if we will evaluate this WFR[2], for classes included in simple UML models, like the model presented in Figure 2, the evaluation result will be fail as you can see in Figure 3.

We suppose that the followings WFRs are straightly connected with the above mentioned WFR. "[1] A Component may only contain other Components in its namespace."

context Component
inv WFR_1:
self.allContents->forAll(c | c.oclIsKindOf(Component))

"[2] A DataType cannot contain any other ModelElements."

context DataType
inv WFR_1:
self.allContents->isEmpty

Taking into account that both Component and DataType are Classifier's descendants the evaluation of these WFR will fail even for the simplest UML models. Analyzing the last three WFR, the following question arises: "What is the role, of these invariants?" In order to eliminate this kind of unpleasant situations, we suggest that in similar cases to describe (using the English language) the meaning of each WFR.

The lasts WFR we will analyze in this paper, are the WFR[4] and WFR[5] defined in the Classifier context.

"[4] The name of an Attribute may not be the same as the name of an opposite AssociationEnd or a ModelElement contained in the Classifier."

context Classifier
inv WFR_4:
self.feature->select(a | a.oclIsKindOf (Attribute))->forAll(a |
not self.allOppositeAssociationEnds->union(self.allContents)->collect(q | q.name)
->includes(a.name))

"[5] The name of an opposite AssociationEnd may not be the same as the name of an Attribute or a ModelElement contained in the Classifier."

context Classifier
inv WFR_5:
self.oppositeAssociationEnds->forAll(o | **not** self.allAttributes->
union(self.allContents)->collect(q | q.name)->includes(o.name))

Even a brief inspection shows us an abusive use of allContents AO. Our statement is based on the fact the WFR do not have to forbid legal situations accepted in object-oriented programming languages like those showed in Figures 4 & 5. More, the last two WFRs, offer us the possibility to highlight the importance of an explicit rule for naming features in UML. This rules have to state explicit that in descendants, the features names are formed prefixing the name feature with "NamespaceName::" for example, if we define in the class A an attribute named "a1", in the class B, this attribute will be called "A::a1". We supposed that the class A was defined in the same package with the class B. If the class A was defined in another package P2, in the class B, the attribute a1 inherited from the class A will be called "P1::A::a1". If in the class B, there are not other attributes named a1, than the attribute defined in the class A can receive the alias a1. Taking into account that a possible name conflict among the attributes defined in a class and the associationEnds attached to the associations connecting this class, can appear only in the execution model (obtained by translating in code the UML design model), the two above mentioned WFR, can be joined in the following WFR:

context Classifier
inv WFR_4:
self.feature->select(a | a.oclIsKindOf (Attribute))->forAll(a |
not self.oppositeAssociationEnds.name->includes(a.name))

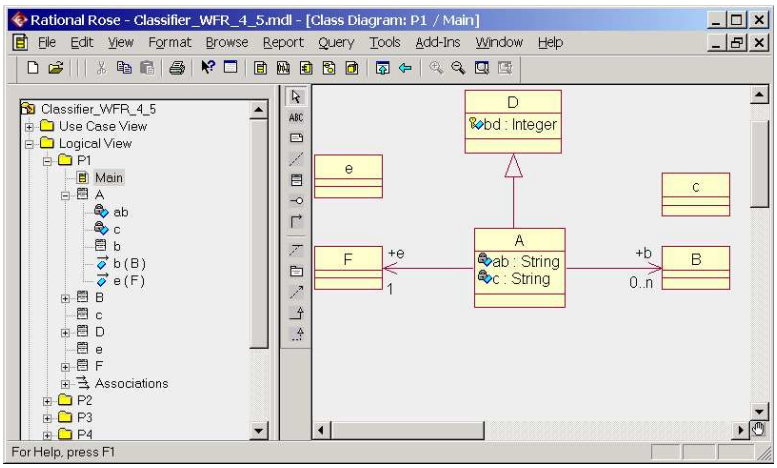

FIGURE 1. The UML model

In order to illustrate the above-mentioned statements, we will evaluate the Classifier WFR [4] and [5]: "The name of an Attribute may not be the same as the name

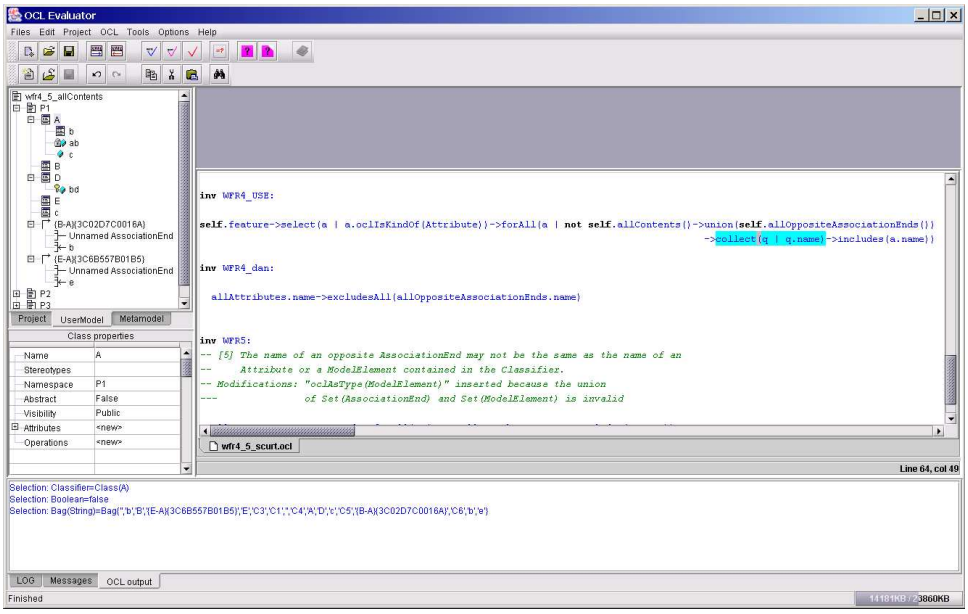

FIGURE 2. WFR_4 Evaluation in the context of class A

of an opposite AssociationEnd (or a ModelElement contained in the Classifier)[1]" using two different formalisms: Rose Script language and OCL.

This because the above-mentioned WFR can be expressed using the Rational Rose Script Language (see next section). If we try to specify (using the Rose Script Language) the Stereotype, Interface, Component, Collaboration WFR, this will not be possible due to the fact that the REI[2] dose not offer access to their required information.

In the case of "access violation", the situation is somewhat similar. As mentioned in [Moore2000]: "An access violation occurs when a class in one package references a class in another package in the absence of an import relationship between the two packages. An access violation will also occur when a package references a class from another package, whose export control is set to Protected, Private, or Implementation. In such cases, the presence of an import relationship between the two packages has no bearing. All references to implementation classes from different packages are sited as violations. Rose provides a very nice GUI for these kinds of inconsistencies – Report: Show Access Violations shows a

[1]The brackets are used to highlight that the inclusion of ModelElements different from attributes is incorrect, as we shall prove in Section 5

[2]Acronym for Rational Rose Extensibility Interface

list of violations created by generalizations, "realize" relationships, dependencies, and "instantiates" relationships. **But it does not show violations created by association classes or types**."

In order to explain the above-mentioned Rose function (menu) limits, let us consider two different packages P1 and P2, placed at the same level in the model. Let's define class A in package P1, and class B in package P2. Set the visibility of A in package P1 as being private and assume that B uses A. Irrespective of the existence of a dependency relationship between P2 and P1, the function "Report Access Violation" doesn't mention this error. Moreover, all the WFR that use the Permission relationship cannot be expressed in Rose Script Language because Rose does not implement this UML relation.

Consequently, the Rose Script Language supports evaluation only of some WFR. Unfortunately even in cases when the WFR specification can be implemented in the script language, it is inefficient as compared to its OCL specification.

6. WFR evaluation using Rose Script Language

The OCL Tools mentioned at `http://www.klasse.nl/ocl`, offer a little part of the functionalities needed to check UML Models. Consequently, in UML CASE tools, checks are usually done using different script languages. The model information accessible by means of script languages is arbitrarilydetermined by each tool provider.

The typical example is offered by Rational Rose, the UML most widely used and known CASE Tool. Consequently, in order to compare the script language support against OCL language support in checking UML models, we wrote the following script, used to check the UML Classifier WFR [4&5] in Rose.

```
'--------------------------------------------------------------------------
'Classifier_WFR_[4&5]
'Verifies the above Well-Formedness Rule
'Description:
'[4] The Name of an Attribute may Not be the same As the Name of an opposite
'AssociationEnd.
'OCL Expression:
'    allAttributes.name->excludesAll(allOppositeAssociationEnds.name)
'--------------------------------------------------------------------------
Sub AllNonPrivateAttributes(theClass As Class,
                            ByRef resultAttrs As AttributeCollection)
        'add the non private attributes of the current class
        Dim attrs As New AttributeCollection
        Set attrs = theClass.Attributes
        For i = 1 To attrs.Count
                If (attrs.GetAt(i).ExportControl <> rsPrivateAccess) And
                   (Not resultAttrs.Exists(attrs.GetAt(i)))
                    Then resultAttrs.add attrs.GetAt(i)
```

```
                End If
        Next i
        'now the attributes of the superclasses
        Dim sc As ClassCollection
        Set sc = theClass.GetSuperclasses
        If sc.Count > 0 Then
                For i = 1 To sc.Count
                        Call AllNonPrivateAttributes(sc.GetAt(i), resultAttrs)
                Next i
        End If
End Sub

Sub AllOppositeAssociationEnds(theClass As Class,
                               ByRef resultAssocEnds As RoleCollection)
        'add the opposite association ends of the current class
        Dim assocEnds As New AssociationCollection
        Set assocEnds = theClass.GetAssociations
        For i = 1 To assocEnds.Count
          If Not
    resultAssocEnds.Exists(assocEnds.GetAt(i).GetOtherRole(theClass))
  Then resultAssocEnds.add assocEnds.GetAt(i).GetOtherRole(theClass)
          End If
        Next i
        'now the opposite association ends of the superclasses
        Dim sc As ClassCollection
        Set sc = theClass.GetSuperclasses
        If sc.Count > 0 Then
                For i = 1 To sc.Count
                  Call AllOppositeAssociationEnds(sc.GetAt(i), resultAssocEnds)
                Next i
        End If
End Sub

Sub AllAttributes(theClass As Class, ByRef resultAttrs As AttributeCollection)
        Dim theAttrs As AttributeCollection
        'get all non private attributes of this class and it's superclasses
        Call AllNonPrivateAttributes(theClass, resultAttrs)
        Set theAttrs = theClass.Attributes
        'add the private attributes of the current class
        For i = 1 To theAttrs.Count
                If (theAttrs.GetAt(i).ExportControl = rsPrivateAccess) Then
                    resultAttrs.add theAttrs.GetAt(i)
                Next i
End Sub
```

```
Function checkWFR(theClass As Class) As Boolean
        Print "Class: " & theClass.Name
        'Get allAttributes
        Dim allAttrs As New AttributeCollection
        Call AllAttributes(theClass, allAttrs)
        Print "Attributes: "
        'print allAttributes - additional operation result;
        For i = 1 To allAttrs.Count
                Print ,allAttrs.GetAt(i).name
        Next i
        'Get allOppositeAssociationEnds
        Dim allOpEnds As New RoleCollection
        Call AllOppositeAssociationEnds(theClass, allOpEnds)
        'print allOppositeAssociationEnds - additional operation result;
        Print "Association Ends: "
        For i = 1 To allOpEnds.Count
                Print ,allOpEnds.GetAt(i).name
        Next i
  'compare the names of the attributes and of the opposite association ends
        Print
        For i = 1 To allAttrs.Count
                For j = 1 To allOpEnds.Count
                        If allAttrs.GetAt(i).name = allOpEnds.GetAt(j).name Then
                                Print "WFR[4&5] Failed !"
                                Print ,"Attribute '" &  allAttrs.GetAt(i).name &
                                       "' = Association End '" &
                                       allOpEnds.GetAt(j).name & "'"
                                checkWFR = false
                                Exit Function
                        End If
                Next j
        Next i
        Print "WFR[4&5] Passed ! "
        checkWFR = True
End Function

Sub Main
        Dim selectedClasses As ClassCollection
        Set selectedClasses = RoseApp.Currentmodel.GetSelectedClasses()
        'check the rule against the selected classes
        If selectedClasses.Count = 0 Then
                MsgBox "No classes selected!"
                Exit Sub
        End If
        'open the viewport and print the results
```

```
        Viewport.Open
        For i = 1 To selectedClasses.Count
                Print "-------------------------------------------------"
                Dim aClass As Class
                Dim ruleResult As Boolean
                Set aClass = selectedClasses.GetAt(i)
                ruleResult = checkWFR(aClass)
                Print "-------------------------------------------------"
                Print
        Next i
End Sub
```

Evaluating the above script, for the UML Model presented in Figure 2, when class A is selected, we will obtain:

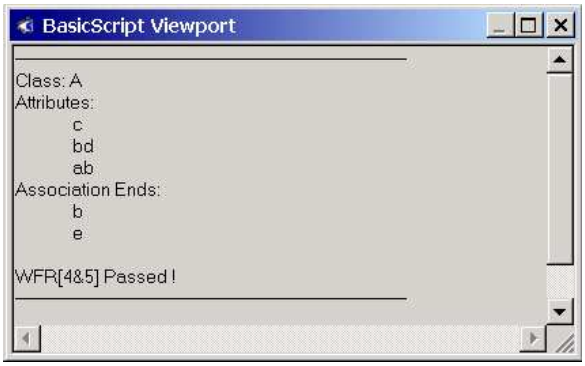

FIGURE 3. Script Evaluation Result

As we can notice, the result is correct. Even if we take into account just the "Function checkWFR(theClass As Class) As Boolean" (considering that the information furnished by subroutines is offered by REI), we notice that the WFR OCL specification (see Section 5) is by far more compact and clear than the above WFR script code.

As mentioned in the previous section, using script languages has some drawbacks. First of all, we cannot express all the WFR specified in UML. More, REI doesn't offer all the information accessible by means of AO. Even in cases when the specification is possible by means of script language, this proved to be cumbersome.

Of course, the WFR semantic correctness represents the precondition which needs to be fulfilled in order to do the above-mentioned checks. This "precondition" cannot be evaluated in the absence of adequate OCL Tool support. This doesn't have to be restrained to the syntactic and semantic WFR checking. It is

mandatory that the OCL tools support the full evaluation of the OCL expressions expressed at the metamodel level. In the next section, we will present the results obtained using our OCL evaluator.

7. WFR evaluation using OCL

In the NEPTUNE IST 1999-20017 Research Project framework, we designed and implemented a tool having as first goal UML model checking. In order to support the tool's independence with respect to UML CASE tools, two main decisions were taken: to use the exchange format for UML models (XMI) and OCL as the rule language. The tool takes the UML models saved in XMI format and checks their correctness against WFR. Compared with USE tool and experience presented in [Richters2000] our tool supports complete UML model checking, enabling the user to take into account all the WFR. (The WFR and AO semantic errors reported in [Richters2000] are type-checking errors). The errors discussed below are "design" errors.

In order to provide the opportunity to do some comparisons between the Rose Script Language and OCL, we will analyze the Classifier WFR [4] and [5].

[4] The name of an Attribute may not be the same as the name of an opposite AssociationEnd or a ModelElement contained in the Classifier.

```
self.feature->select(a | a.oclIsKindOf(Attribute))->forAll(a |
not self.allOppositeAssociationEnds->union(self.allContents)->collect(q | q.name)
->includes(a.name))
```

[5] The name of an opposite AssociationEnd may not be the same as the name of an Attribute or a ModelElement contained in the Classifier.

```
self.oppositeAssociationEnds->forAll(o | not self.allAttributes->
union (self.allContents)->collect(q | q.name )->includes(o.name))
```

A simple analysis of these two rules gives us the opportunity to notice their similarity. From the informal point of view, in the description made in natural language only the phrase topic is changed. Concerning the OCL specification, in [4] only the attributes defined in the Classifier are taken into account. On the contrary, in [5] the attributes defined in the ancestors are also included. Our opinion is that this second solution is correct. However some corrections have to be made.

First of all, both in WFR[4] and [5], as mentioned in [Richters2000], we notice that from the type checker point of view, the expression self.allAttributes->union(self.allContents) is erroneous because we try to join a Set(Attributes) with a Set(ModelElements). In order to solve this semantic error, the solution proposed at `http://www.db.informatik.uni-bremen.de/~mr` is to change the order of operation. A more careful analysis of the above WFR shows us that the OCL expressions contain redundancies. The expression:

```
forAll(a | not self.allContents->union(self.allAttributes)->collect(q | q.name)->
includes(a.name))
```

is equivalent with:

forAll(a | **not** allAttributes.name->union(allContents.name) ->includes(a.name)).

Now the WFR is semantically correct and can be evaluated. For the UML model presented in Figure 2, the evaluation will fail (see Figure 3). Analyzing Figure 3, we find the reason. The Boolean attribute b, declared in class C1, is included in the set allAttributes, computed for class A, but the visibility of b is private, so it shouldn't be included in that set. As we can see, in Figure 4, this mistake was discarded; removing all private attributes defined in the classifier's ancestors. In fact the correct strategy is to do this correction in the Classifier's allFeatures() AO. Taking into account its usage, this represents a very important correction in the AOs. The change will be automatically propagated in allAttributes() and allOperations().

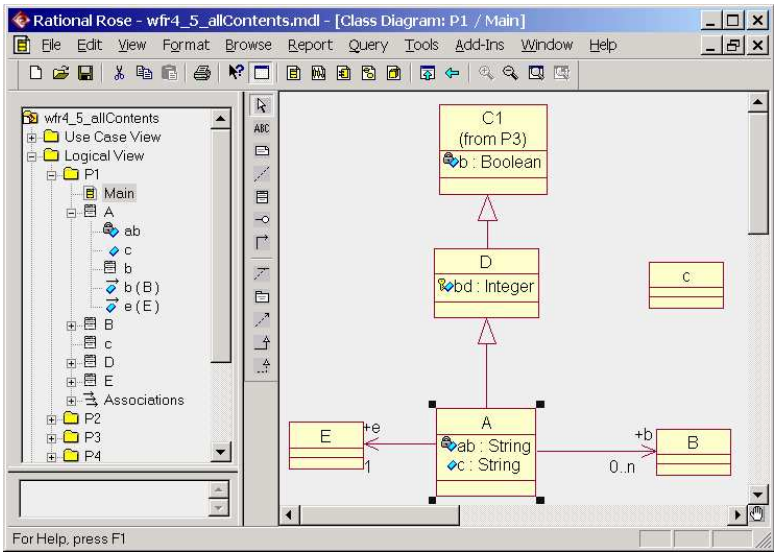

FIGURE 4. Another UML model

Solving the above-mentioned problem, another error will occur. This new error is determined by the reunion allAttributes.name->union(allContents.name). As you can see, the set allContents.name includes 'b', the name of the inner class declared in class A(see Figure 2 – the Rational Rose browser). As a consequence, the WFR evaluation will fail due to the name conflict between the associationEnd b and the inner class b. The above reunion is erroneous because it forbids modeling legal situations found in programming languages such as Java, C++, etc. Moreover, oppositeAssociationEnds has to be replaced by allOppositeAssociationEnds because,

due to inheritance, the associationEnds declared in the classifier's ancestors are accessible.

Taking into account all the above suggested corrections, we can now write the Classifier's WFR[4]:

allOppositeAssociationEnds.name->excludesAll(allAttributes.name).

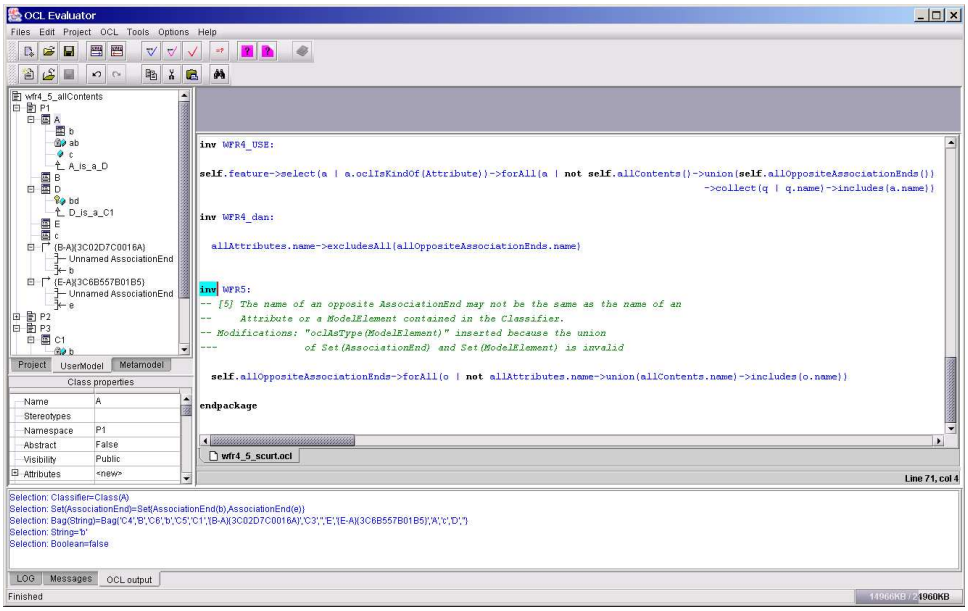

FIGURE 5. WFR [5] Evaluation

This last proposed specification covers equally the initial Classifier's WFR[4] and WFR[5]. It is even more concise and clear as compared to the textual specification and can be directly evaluated; a big advantage.

In order to understand the use of our OCL evaluator, we mention that the UML 1.4 metamodel is automatically loaded every time when the evaluator is launched. The next mandatory activities are the loading of the OCL file and the UML user model. Their order does not matter. The third mandatory activity is the OCL file (expression) compilation, followed by the context specification. In Figures 3 and 4, the first two lines of the output pane mark successful compilation. In the third line, successful loading of the UML model is signaled. Finally in the fourth line, the context specification is noticed. In both the above-mentioned figures, the last output pane line shows the results of the invariants evaluation. In Figure 3 the lines 5-8 illustrate the evaluation of oppositeAssociationEnds, allAttributes.name, allContents.name allAttributes.name->union(allContents.name), respectively. In Figure 4,

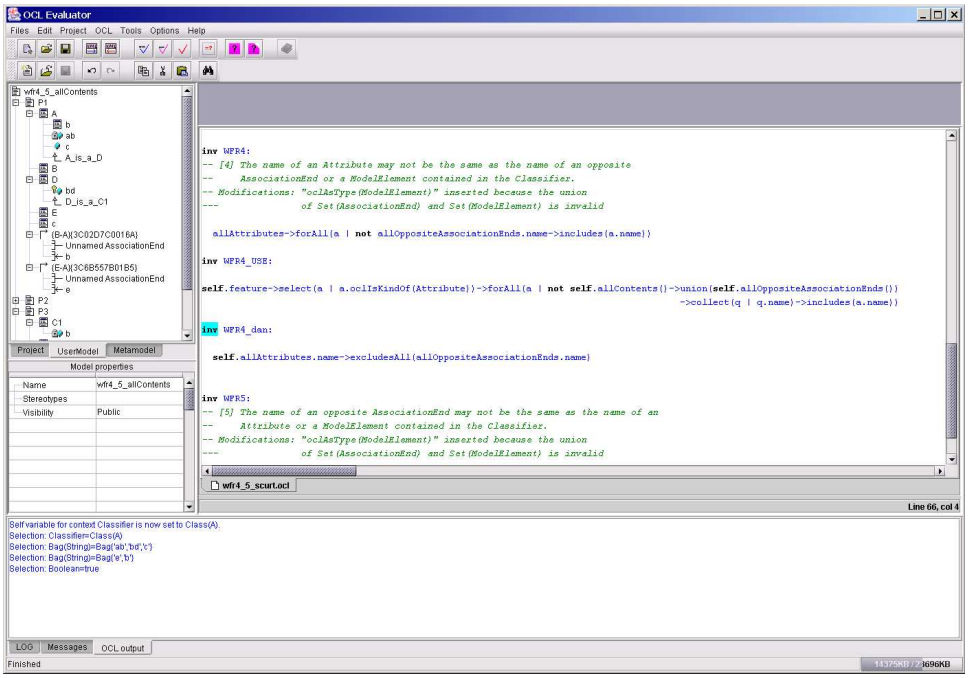

FIGURE 6. Proposed WFR [4&5] Evaluation

the lines 5-6 show the evaluation result for: allOppositeAssociationEnds.name and allAttributes.name. The possibility to evaluate different specification "chunks" is very useful in debugging OCL expressions.

8. Conclusions and future work

In this paper we tried to analyze the state of the art in the UML model-checking domain. Our research demonstrated that today, for this activity, the appropriate formalism should be by far OCL. Using script languages is also possible. As we showed, this last formalism has different drawbacks. The experience we acquired in the NEPTUNE Research Project showed that in order to be efficient, the OCL tools have to support the evaluation of expressions. The syntactic and semantic errors like those mentioned in other papers are not sufficient. Moreover, the access to the UML metamodel is mandatory. Evaluating UML 1.4 AO and WFR gives us the opportunity to find a lot of conceptual errors. The space of this paper does not allow us to mention all of them. Equally important, we identified redundant OCL specifications and some WFR not yet mentioned in the UML specification. Specifying AO and WFR has many solutions. It is very important to find the

simplest and clearest ones. In order to check UML models, the AO and WFR completeness and correctness represent a precondition. Consequently, our results offer support for the accomplishment of this "precondition". At least, as far as we are concerned, we don't know similar results published in other scientific papers. We are now trying to identify all the errors in the AO and the WFR and to describe solutions for all of them.

References

[Moors2000] Michael Moors, Consistency Checking – Rose Architect, Spring Issue, April 2000, http://www.therationaledge.com/rosearchitect/mag/ index.html

[Richters 2000] Mark Richters, Martin Gogolla, Validating UML Models and OCL Constraints, "Proc. 3rd International Conference on the Unified Modeling Language (UML)", Springer-Verlag, 2000

[Chiorean01] Dan Chiorean, Using OCL beyond specification, Lecture Notes in Informatics 7, "Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists, 2001, pag. 57-69

[UML 1.4] UML 1.4 Draft Specification, February 2001, `http://uml.sh.com`

[Warmer1999] Warmer J, Kleppe A., "The Object Constraint Language", Addison Wesley, 1999

[DresdenOCL] `http://dresden-ocl.sourceforge.net/index.html`

Babeş-Bolyai University, Computer Science Research Laboratory, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania

E-mail address: `chiorean@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

DIAGRAM DESIGN IN OCL EVALUATOR

HORIA CHIOREAN

Abstract. OCL Evaluator is a tool developed by the Computer Research Laboratory of the Babes-Bolyai University (LCI), designed for checking UML models stored in XMI format. The main purpose of this software is that of checking (verifying). The general use case scenario is: the user designs his application using a case tool like Rational Rose, Together, etc. Once this design faze is finished, the user has a model for the problem. This model is exported using the same case tool or some other program into an XMI document. The XMI document is inputted into our tool and verified. These verifications are done according to certain rules, that can be very complex and are grouped into several categories.

On the other hand, once errors are reported for a model, it's natural to allow the user to correct these errors. This is where the diagrams "kick in". Diagrams can simplify very much the process of error correcting, because they can provide a visual representation for a model.

The problem was to implement these diagrams in OCL Evaluator, using the Java programming language.

1. Achievements regarding OCL

The Object Constraint Language (or OCL) first appeared in 1997 as part of UML's 1.1 specification. OCL is a formal language used to express constraints. These typically specify invariant conditions that must be satisfied by the system being modeled.

Although it has been developed some years ago, very little support has been given to OCL in the sense that there are very few software tools that give the user the possibility to check his/hers model using the object constraint language. Well known CASE tools, such as Rational Rose, have plugins that allow certain verifications to be done using OCL, but these plugins are by no means enough in order to use OCL's the full power.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

2. The importance of UML diagrams

A diagram (in general) could be considered a graphical representation of certain elements together with their relations. The reason why diagrams are so important is because they provide a graphical representation for a system (or for a part of the system). Having such a representation, makes it a lot easier to understand how that system works.

The Unified Modeling Language (or UML) is, as its name states, a modeling language. In other words, it is a language that is used to design models for problems. (by problems, we mean software problems, that can be solved using an Object Oriented approach) It is natural for such a language to have diagrams in its specification and therefore UML (in the 1.4 specification), defines the following kinds of diagrams:

(1) Static Structure Diagrams – class diagram and object diagram.
(2) Use Case Diagrams – the use case diagram.
(3) Interaction Diagrams – collaboration diagram and sequence diagram.
(4) State Charts Diagrams – the state chart diagram.
(5) Activity Diagrams – the activity diagram.
(6) Implementation diagrams – the component diagram and the deployment diagram.

Each of these diagrams contains several elements and relations, according to their type, that are abstract elements defined in UML's specification. (elements like classes, actors, use cases, messages, objects, associations, etc) Each and every one of those, has a graphical notation like: rectangles for classes, lines for associations, ovals for use cases, etc, graphical notations that are used in a diagram.

The diagrams should give the user the possibility of representing only parts of the model or the model in it's entirety. This means that a diagram should have the following features:

(1) To allow one or more elements to be represented in more then one diagram. In other words, an element can be represented in multiple diagrams.
(2) Deleting one/more elements from a diagram, without affecting the model.
(3) Undo/Redo functionalities.
(4) Element filtering – this is a very important feature, because it allows the user to see only the part of that element that he is interested in. Plus it can simplify a great deal the situation when you have very large diagrams.

Most of the well known CASE tools: Rational Rose, Together, Use, etc. support diagrams, but each has their drawbacks.

3. Our solution: UML diagram design in OCL Evaluator using JGraph

We developed the only tool, OCL Evaluator, that is based on the object constraint language, with the sole purpose of providing efficient support for model checking. Although there are several OCL compilers out there (like the Dresden OCL Compiler or IBM's OCL Compiler), none provide an adequate user interface and are therefore very difficult to use. OCL Evaluator is based on our own OCL compiler, but also has an extensive user interface that facilitates the process of checking. Moreover, the user not only has the possibility of checking a system, but also to correct that system based on eventual errors.

In this context, it was decided that to include a graphical representation of a system, by means of diagrams, was very important because firstly, it would facilitate the checking process a great deal by giving a visual representation and secondly, no other OCL software had this facility.

Although only class diagrams and use case diagrams have been implemented, most of these diagrams have a thing in common: they can be looked at similar to a graph. In other words, the structure of a diagram is similar to that of a graph where the objects are vertices (cells) and the relations between them are edges.

Therefore, when it was decided to include diagrams in the OCL Evaluator, there were 2 options: either to implement a graphical library from scratch or to use an existing graph library and to modify it so that it would fulfill the need of representing a good UML diagram. The later was chosen in the end and the graph library chosen was JGraph (`http://www.jgraph.com`) .

JGraph is a freeware, Java based library, used to represent graphs. The intention behind it, is to provide a freely available and fully Swing compliant implementation of a graph component. As a Swing component for graphs, JGraph is based on the mathematical theory of networks, called graph theory, and the Swing user interface library, which defines the architecture. By combining these two, JGraph becomes a Swing user interface component used for visualizing graphs.

The design of the JGraph is similar to that of a Swing component. In other words, besides the fact that JGraph **is** a Swing component (because it subclasses JComponent), its architecture is based on Swing Model View Controller (or MVC).

Figure 1 shows, according to [1], the diagram of a JComponent (JTree), which shows the way in which the MVC pattern is applied.

On the diagram, you clearly see all the participants (except the controller): the component itself - JTree, its UI (which has the responsibility of rendering the component) - TreeUI, and its model which encapsulates all the information – DefaultTreeModel. The control in Swing MVC is encapsulated in the UI-delegate, which is in charge of rendering the component in platform-specific manner, and mapping the events from the user interface to transactions, that are executed on the model.

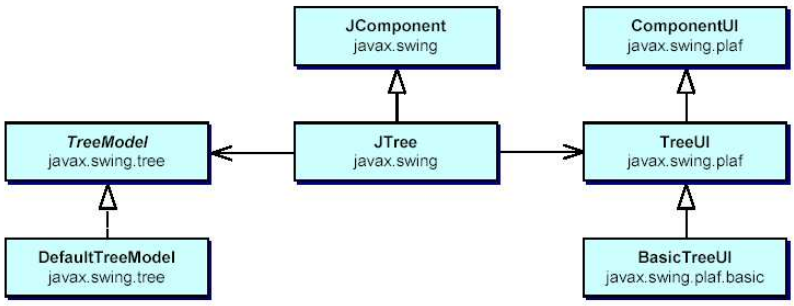

FIGURE 1. JTree MVC architecture

The JGraph component inherits this basic setup from the JComponent class and its UIdelegate, which implements the ComponentUI abstract class.

As in the case of text components, the split between platform-dependent and non-platform Design and Implementation of JGraph dependent attributes, is implemented using the concept of views, which are independent from the elements that appear in the model.

In Swing's text components, the elements of the model implement the Element interface, and for each of these elements, there exists exactly one view. These views, which implement the View interface, are either accessed through a mapping between the elements and the views, or through an entry point called root view, which is referenced from the text component's UIdelegate.

JGraph has an analogous setup, with the only difference that a graph view is referenced by the JGraph instance. The cells of the graph model implement the GraphCell interface, which is JGraph's analogy to the Element interface. The cell views implement the CellView interface, in analogy to Swing's View interface. The cell views are either accessed through the CellMapper mechanism, or through the graph view, which is an instance of the GraphView class. However, since the GraphView class works together with other classes, the analogy with Swing's text components is more helpful to understand the separation between the cell and the view.

In contrast to text components, where the geometric attributes are stored in the model only, JGraph allows to store such attributes separately in each view, thus allowing a graph model to have multiple graphic configurations, namely one for each attached view.

Figure 2 shows JGraphs's model view controller diagram, according to [1].

The key elements in a JGraph are the GraphCells. These cells are inserted into the GraphModel and using the GraphUI they are given a visual representation. There a 3 kinds of GraphCells: Vertexes, Edges and Ports. While vertexes and

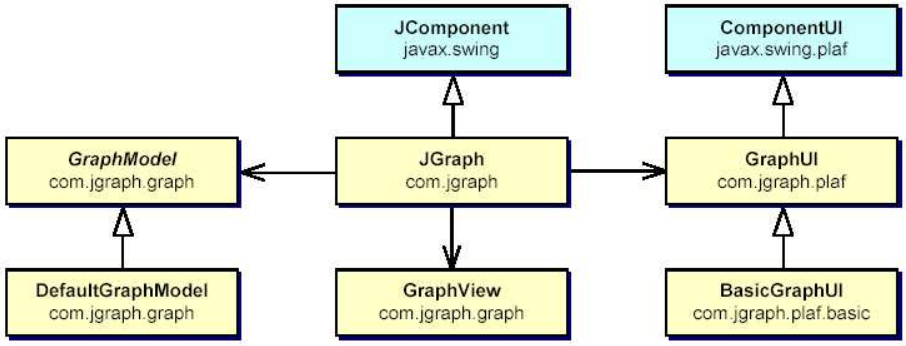

FIGURE 2. JGraph MVC

edges are the common elements of a graph, the port is new concept. In JGraph, a port is connection point for an edge. Ports are added as children to vertexes, providing a way for an edge to connect to a vertex. The graphical representation of a GraphCell is achieved using CellView. CellView is the base class for all the special views such as: VertexView, PortView and EdgeView.

The graph cell has 2 very important attributes: it holds a reference to an Object (referred to as the user object) and it has a corresponding View. This view is in fact an instance of the JGraph's CellView interface and it also holds a reference to a Renderer (which normally is a subclass of JComponent). This renderer is responsible for the painting. For each cell in the graph model, there exists exactly one cell view in each graph view, which has its own internal representation of the graph model. The renderers are instantiated and referenced by the cell views.

Renderers are based on the idea of the TreeCellRenderer class from Swing, and on the Flyweight design pattern. The basic idea behind this pattern is to "use sharing to support large numbers of fine-grained objects efficiently."

Because having a separate component for each CellView-instance would be expensive, the component is shared among all cell views of a certain class. A cell view therefore only stores the state of the component (such as the color, size etc.), whereas the renderer holds the component's painting code (for example a JLabel instance – in the case of Vertex).

The CellViews are painted by configuring the renderer, and painting the latter to a CellRendererPane , which may be used to paint components without the need to add them, as in the case of a container. The renderers in JGraph are used in analogy to the renderer in JTree, just that JGraph provides more than one renderer, namely one for each type of cell. Thus, JGraph provides a renderer for vertices, one for edges, and one for ports. For each subtype of the CellView

interface, by overriding the getRenderer method, you may associate a new renderer. The renderer should be static to allow it to be shared among multiple instances of a class.

The renderer itself is typically an instance of the JComponent class, with an overridden paint method that paints the cell, based on the attributes of the passed-in cell view. The renderer also implements the **CellViewRenderer** interface, which provides the getRendererComponent method to configure the renderer. According to [1], Figure 3 shows the architecture of the renderers.

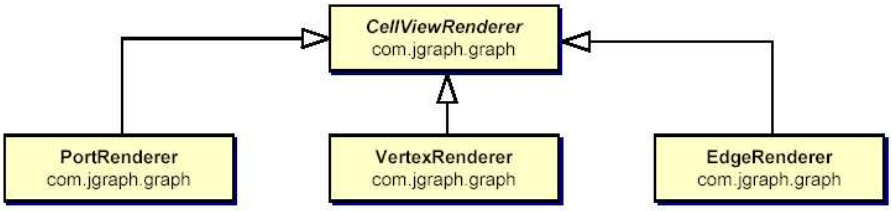

FIGURE 3. JGraph's renderers

The three default implementations of the CellViewRenderer interface are the VertexRenderer, EdgeRenderer and PortRenderer classes.

4. Our solution

There were 2 main challenges in using this library: firstly fixing some bugs that were present in the implementation of JGraph (the most important of which being an annoying flicker when dragging cells) and secondly, adapting JGraph so that it could be used for UML diagrams.

There were two main bugs in JGraph 1.0:

(1) When dragging a cell or an edge, or when changing the size of a cell, there would be a visible and annoying flicker on the screen.
(2) When trying to bend an edge, the connection point would not be inserted correctly. (bending edge is achieved by adding connection points to an edge and dragging those points).

Fixes:

(1) The overlay() method in the BasicGraphUI class did not use Swing's double buffering technique, this begin the reason for the flicker. The method was modified so that the painting would be done in an off-screen buffer and only after that displayed on screen.
(2) The OnMouseClick() method in the EdgeHandler class did not make correctly the calculations about the location of the control point.

In the current implementation of the Evaluator, the diagrams were implemented graphically by using the JDektopPane and JInterFrame Swing classes. This meant that a diagram would contain an instance of a JGraph and this instance would actually work as a canvas for JInternalFrame. This design allows (and so in should) the existence of multiple diagrams.

Figure 4 shows what the diagrams look like (when they are empty).

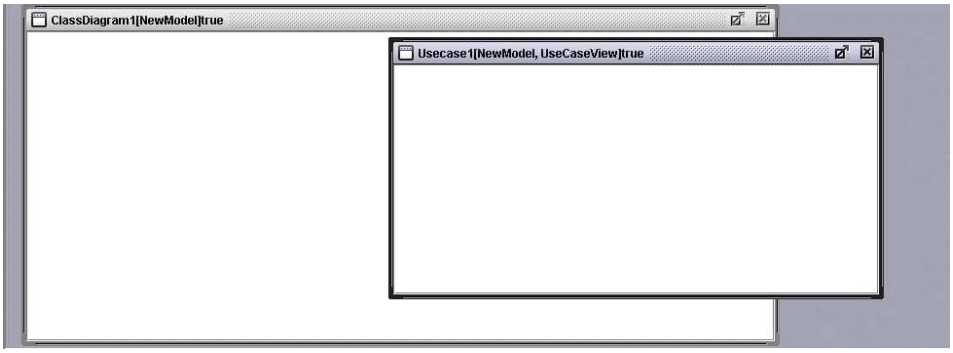

FIGURE 4. Diagram Overview in OCL Evaluator

Each element in a diagram, would have to be a cell in a graph. However, because JGraph's support for cells was limited to only one kind of cell - the Vertex, which is a rectangle with an optional text in the center, the cells implemented by us were: Class Cell, Package Cell, Actor Cell and UseCase cell.

Because JGraph uses the Factory method to create views for each type of cell, (this is achieved trough a method called createView in the JGraph class), we sub-classed JGraph and created our own DiagramGraph. This class represents the graph behind the diagram and it's responsible for creating the correct views for each kind of cell. We didn't implement a new GraphModel because the one provided by the library (DefaultGraphModel) was general enough for what we needed.

Therefore, we implemented only the necessary types of cells. Each one of Class Cell, Use Case Cell, Package Cell and Actor Cell represents the graph cell, and holds the user object.

Every custom cell, also has a corresponding view related to it, as shown in Figure 6:

Every renderer, has an appropriate paint() method. This method will be invoked by BasicGraphUI, when the rendering mechanism takes place. This is the place where we wrote the code that displays each cells according to its abstract counterpart from UML 1.4. The cell class holds a reference to an object for the model (the object which it represents). So when the paint() method is invoked by

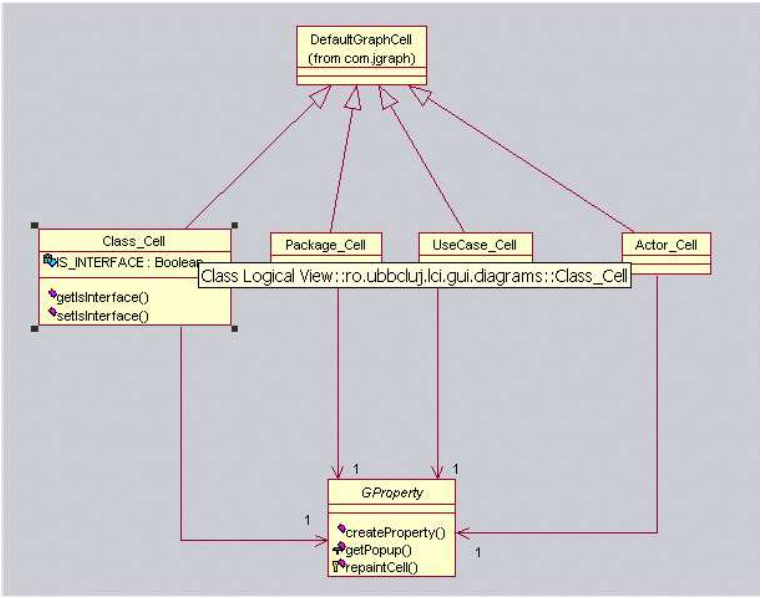

FIGURE 5. Diagram Cells implemented for OCL Evaluator

the view, the user object is accessed, and the painting is done according to the information extracted from this object. For example, in order to draw a class, before drawing it's outline (the rectangle), we take the name of the class, its attributes and methods from the object (which must be an instanceof Classifier), and draw them using the drawString() and drawImage() methods from java.awt.Graphics. This painting mechanism is used for every kind of cell, the only difference being the user object and the shape of the cell (shape which conforms to the UML 1.4 specification).

In addition to the having a view, we implemented two other classes for our cells. The first one called GProperty and encapsulates the graphical property for that cell – meaning the fill color, the outline color, the title font size, the body font size, etc. The second one is called AbstractFilter and represents a filter. This filter has been only implemented for ClassCell, and it allows the filtering of classes according to the visibility of the attributes and methods.

Holding a model element as a user object, provides greater flexibility in the sense that when this user object, which is always an instance of the Element interface from UML1.4, is modified, a simple repaint is enough to visualize the change in all the corresponding diagrams.

As far as the relations between elements are concerned, in any kind of UML diagram, these relations are represented by straight lines with possible decorations

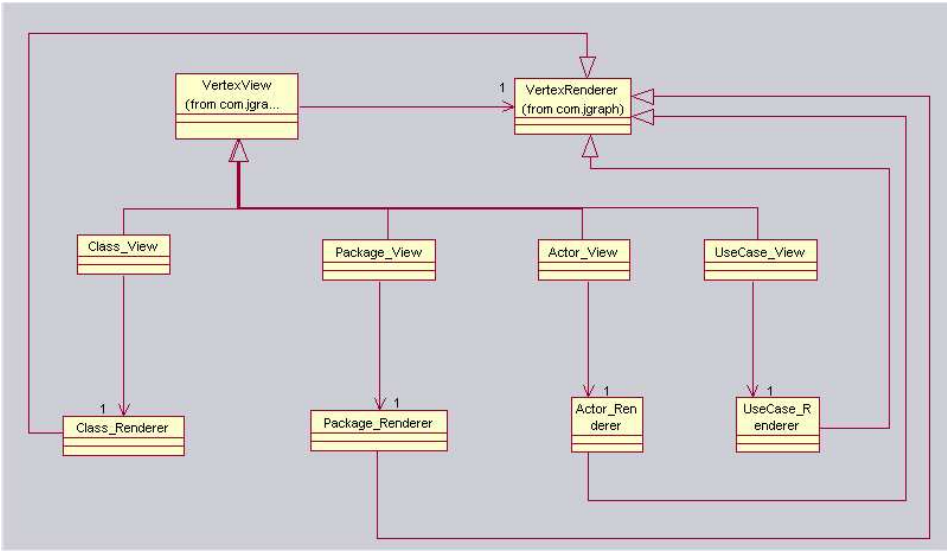

FIGURE 6. Custom Views and Renderers

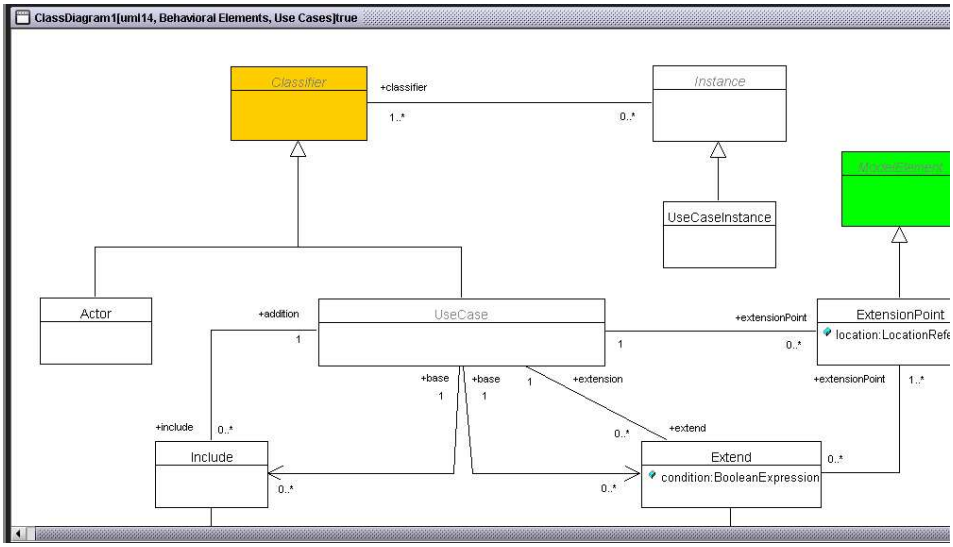

FIGURE 7. A class diagram

at the extremities. Already JGraph already had the support for this, there were two important issues that were solved:

(1) An Association in UML, besides being represented as a straight line, also has several additional text labels that indicate the role names and the multiplicity of that association. Therefore we implemented text labels that could be set at the end of the edge. Although these labels are not tied to the edge, by double clicking an edge, they will gather around that edge.
(2) See Figure 8.

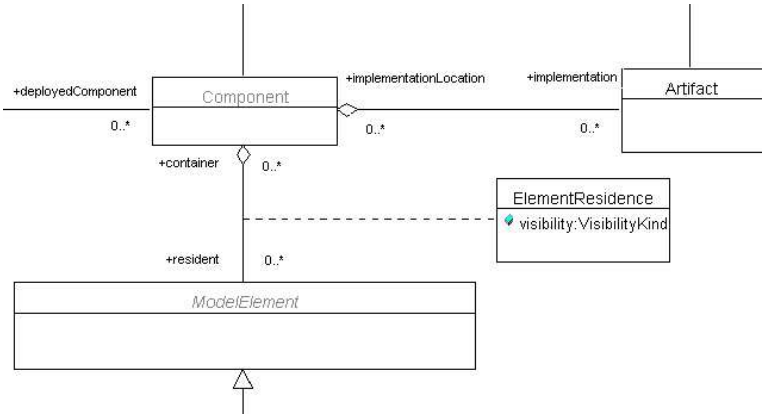

FIGURE 8. Diagram with an Association Class

An Association Class in UML is represented with a dashed line that connects a class with an association. This meant that we had to connect a cell with an edge using a second edge. We stated previously that the key to connecting edges are the ports. These ports act as a sort of "glue" in the sense that they keep edges connected. However, in the JGraph library the only cells that are allowed to have ports are the vertexes.

So, we needed some sort of hybrid edge that allowed the addition of ports as children on one hand, but would still have the capability to connect to vertexes together. We solved this problem by creating 2 special classes (with their corresponding views): SpecialEdge and SpecialPort, which extend DefaultEdge respectively DefaultPort. However, SpecialEdge's view class - SpecialEdgeView is similar to the view of a vertex making SpecialEdge therefore both an edge and a vertex. Figure 9 illustrates this.

The OCL evaluator is divided in 5 major parts, as shown in Figure 10.

You can create a diagram by right-clicking in the browser on the desired parent for the diagram (a package usually), and from the pop-menu selecting New. In order to add elements to the diagram, you can achieve this in 2 ways:

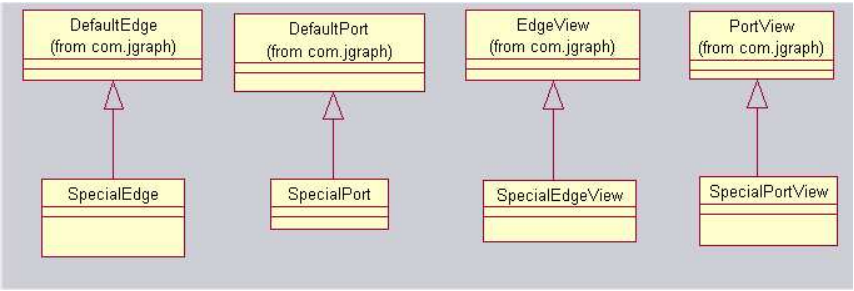

FIGURE 9. The Special Cells

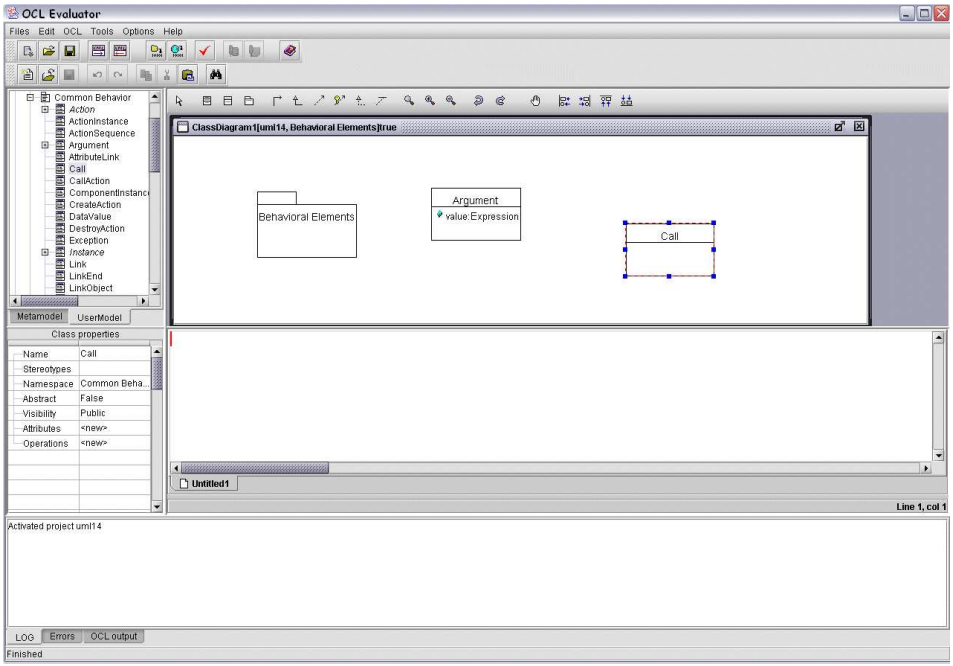

FIGURE 10. OCL Evaluator Overview

(1) Dragging and dropping an element from the browser onto the diagram, in which case the element will be represented graphically according to its information.
(2) Using the toolbar to create new elements/relations.

5. Conclusions and future work

OCL Evaluator is not yet in a release version. It's still in beta version. However, several extensions of our project are possible such as: using OCL to verify components and using OCL for checking XMI documents.

Of course that the two types of diagrams that were implemented, although they are the most "fundamental diagrams", are not enough for a competitive tool. New diagrams will be implemented as the project extends. However, because JGraph's extensibility is limited, a decision will have to be taken whether to continue using this library, or to implement a new one.

References

[1] Alder. The JGraph Paper. `http://jgraph.sourceforge.net/paper.pdf`
[2] Alder. The JGraph 1.0 API Specification. `http://api.jgraph.com`
[3] Gamma, Helm, Johnson, Vlissides. Design Patterns. Addison-Wesley, Reading MA, 1995.
[4] Geary. Graphic Java 2, Volume II: Swing (3rd Edition). Sun Microsystems, Palo Alto CA, 1999.
[5] OMG Unified Modeling Language Specification, Version 1.4 draft, February 2001

Babeş-Bolyai University, Computer Science Research Laboratory, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

A PROPERTY SHEET

SORIN MOLDOVAN

Abstract. The graphical user interfaces (GUI) are very important today. There is no application without designing a friendly, easy to understand and accessible environment, both for beginners and for expert users. The aim of this article is to analyze some aspects specific to graphical interfaces provided by CAD and CASE tools and to offer a solution for the programmers developing components for such a tool in JAVA programming language. This solution is illustrated in the design and implementation of a property sheet used to display general and specific properties for the UML entities the user works with.

1. Why such a component?

The objects with which the user operates are, most of the time, complex and vast, having lots of attributes (properties). For example, Visual Basic users, when they create a form, have to manage, for each graphical component, a large list of properties (name, position, color, font, text, etc.). It would be a good idea if these characteristic attributes of an object were grouped in several categories. At a certain moment, the user is not interested in all the properties, but only in some of them. Therefore, some categories should be summarized, rather than detailed. Through this facility, we can dynamically control the level of detail at which we study the object and also have a custom view upon an object.

Another aspect is related to editing the attribute values together with their validation. By referring again to the example about the Visual Basic forms, we notice that attributes such as name, position, can have an infinite (or at least very large) domain of values. For editing these attributes, we will use controls of type TextField (they may be specialized for alpha-numeric or only numeric values, or may have additional constraints). Some other properties can only have a finite domain of values (sometimes very small – for example, the visibility can be either True or False). In this case, instead of the user providing a value as input and then checking if the value is valid or not, the component will provide a list of values (which will cover the whole domain), and the user will choose one of these

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

possible values. For properties such as the used font, editing is more complex and we will open a dialog (probably a system dialog). Between some objects there are dependency relationships (*object1* "owned by" *object2*, for instance). If the properties window of *object1* has the "Owner" property with a value of *object2*, then in some cases we can navigate the relationship from *object1* towards *object2* through the property sheet.

In the framework of the OCL Evaluator project the need of such property sheet occurred. In addition to the model browser and diagrams which offer a view upon the whole project or a part of it, one may have the possibility to inspect a certain element and to modify its state. It acts as a view and a controller on the UML model. Figure 1 illustrates how class properties are displayed. What this example does not reveal is the possibility to edit each property using a specific control depending of the kind of information the property contains.

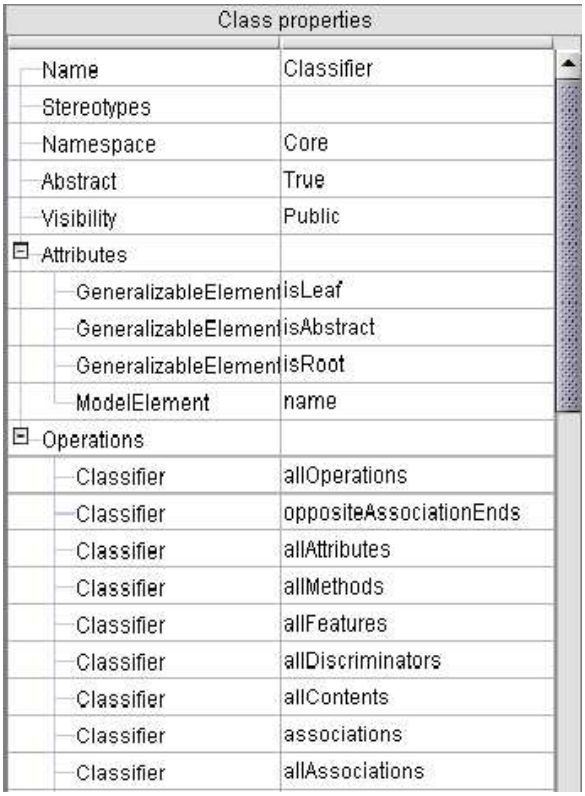

FIGURE 1. The property sheet in a CASE tool

The implementation of the property sheet was realized in JAVA programming language using the Swing package. First of all, we present the main technique we used.

2. Classic Model-View-Controller (MVC) Architecture

The MVC Architecture is designed for applications that need to provide multiple vies of the same data. MVC separates applications into three types of objects:

Model: Maintain data and provide data accessor methods; is the application object.

Views: Paint a visual representation of some or all of a model's data

Controller: Defines the way the user interface reacts to user input.

Models are responsible for maintaining data; for example a notepad application would store the current document's text in a model. Models typically provide methods to access and modify their data. Model also fires events to registered views when a model is changed, and the views respond by updating themselves based on the model change.

Views are responsible for providing a visual representation of some portion of a model's data. For example, a notepad application would provide a view of the current document by displaying some or all of the text stored in the model.

Controllers handle events for views. Swing listeners (such as mouse and action listeners) are MVC controllers. The notepad application mentioned previously would have mouse and key listeners that made changes to the model or view as appropriate.

Before MVC, user interface designs tended to consider these objects together. MVC decouples them to increase flexibility and reuse. MVC is a powerful design for a number of reasons. First, multiple views and controllers can be plugged into a single model, which is the basis for Swing's pluggable look and feel.

Second, a model's views are automatically notified when the model is changed, changing a model property in one view results in subsequent updates of the model's other views.

Third, because model is not dependent upon views, models do not have to be modified to accommodate new types of views and controllers.

We will refer to MVC architecture when we'll describe the interaction between the property sheet and the UML model.

Swing MVC is a specialized version of classic MVC meant to support pluggable look and feel instead of applications in general. Swing lightweight components consist of the following objects:

- a model that maintains a component's data;
- a UI delegate that is a view with listeners for handling events;
- a component that extends JComponent class.

Swing models translate directly to classic MVC models. The components delegate their look and feel to a UI delegate. UI delegates correspond to a view-controller combination in classic MVC. Controllers are referred to as listeners from here on.

Taking into account the Swing MVC architecture we describe how we have create the JTreeTable class used in our view component upon the UML model.

3. JTreeTable

A *TreeTable* is a combination of a Tree and a Table – a component capable of both expanding and contracting rows, as well as showing multiple columns of data. The Swing package does not contain a JTreeTable component, but it is fairly easy to create one by installing a JTree as a renderer for the cells in a JTable.

In Swing, the JTree, JTable, JList, and JComboBox components use a single delegate object called a *cell renderer* to draw their contents. In fact it is the **view** from the MVC pattern. A cell renderer is a component whose *paint()* method is used to draw each item in a list, each node in a tree, or each cell in a table. A cell renderer component can be viewed as a "rubber stamp": it's moved into each cell location using *setBounds()*, and is then drawn with the component's *paint()* method.

By using a component to render cells, you can achieve the effect of displaying a large number of components for the cost of creating just one. By default, the Swing components that employ cell renderers simply use a JLabel, which supports the drawing of simple combinations of text and an icon. To use any Swing component as a cell renderer, all you have to do is create a subclass that implements the appropriate cell renderer interface: TableCellRenderer for JTable, ListCellRenderer for JList, and so on.

4. Rendering in Swing

Here's an example of how you can extend a JCheckBox to act as a renderer in a JTable:

```
public class CheckBoxRenderer extends JCheckBox
      implements TableCellRenderer {
   public Component getTableCellRendererComponent(JTable table,
      Object value, boolean isSelected,
      boolean hasFocus, int row, int column) {
         setSelected(((Boolean)value).booleanValue());
         return this;
      }
}
```

5. How the example program works

The code showed above shows how to use a JTree as a renderer inside a JTable. This is a slightly unusual case because it uses the JTree to paint a single node in each cell of the table rather than painting a complete copy of the tree in each of the cells. We start in the usual way: expanding the JTree into a cell render by extending it to implement the TableCellRenderer interface. To implement the required behavior or a cell renderer, we must arrange for our renderer to paint just the node of the tree that is visible in a particular cell. One simple way to achieve this is to override the *setBounds()* and *paint()* methods, as follows:

```
public class TreeTableCellRenderer extends JTree
      implements TableCellRenderer {
   protected int visibleRow;
   public void setBounds(int x, int y, int w, int h) {
      super.setBounds(x, 0, w, table.getHeight());
   }
   public void paint(Graphics g) {
      g.translate(0, -visibleRow * getRowHeight());
      super.paint(g);
   }
   public Component getTableCellRendererComponent(JTable table,
      object value,
      boolean isSelected,
      boolean hasFocus,
      int row, int column) {
         visibleRow = row;
         return this;
      }
}
```

As each cell is painted, the JTable goes through the usual process of getting the renderer, setting its bounds, and asking it to paint. In this case, though, we record the row number of the cell being painted in an instance variable named *visibleRow*. We also override *setBounds()*, so that the JTree remains the same height as the JTable, despite the JTable's attempts to set its bounds to fit the dimensions of the cell being painted.

To complete this technique we override *paint()*, making use of the stored variable *visibleRow*, an operation that effectively moves the clipping rectangle over the appropriate part of the tree. The result is that the JTree draws just one of its nodes each time the table requests it to paint.

In addition to installing the JTree as a renderer for the cells in the first column, we install the JTree as the editor for these cells also. The effect of this strategy

is the JTable then passes all mouse and keyboard events to this "editor" – thus allowing the tree to expand and contract its nodes as a result of user input.

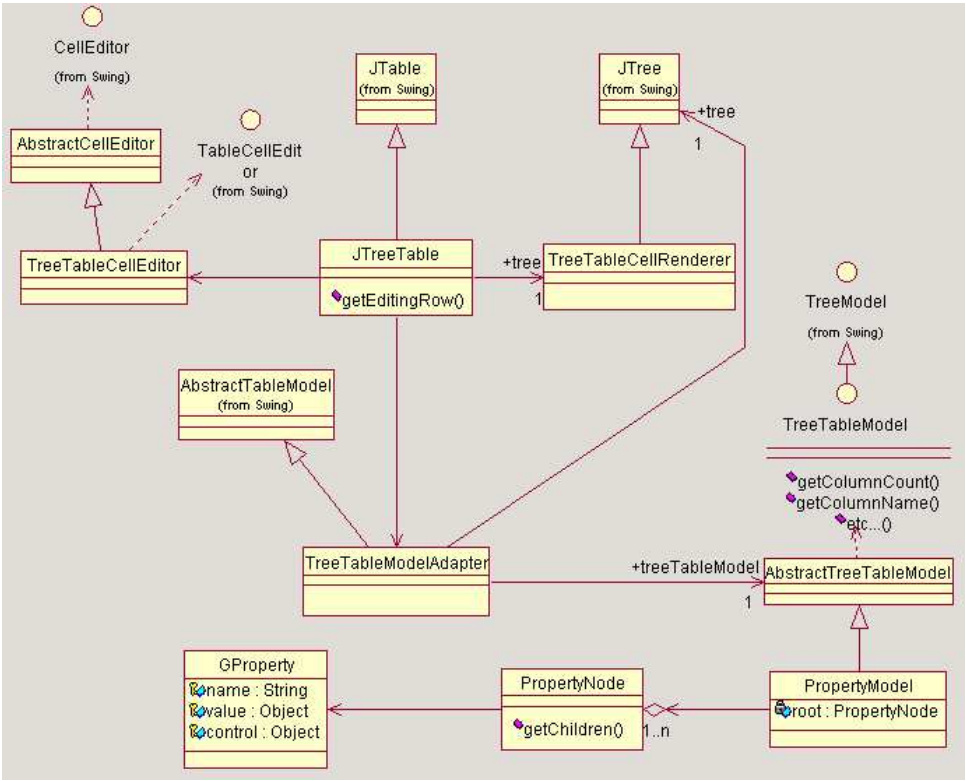

FIGURE 2. TreeTable Architecture

6. A DIFFERENT EDITOR FOR EACH ROW

We mentioned above that this component has a specific editor depending on the values domain range. So we subclass DefaultCellEditor and name it TableCellEditor. This class knows what editor is responsible for editing of a row. A private member *editors* contains in a HashSet all the editors used. It is not necessary for each row to provide an editor. If a row has no editor associatd a default editor is provided.

We add a method to this class: *selectEditor(MouseEvent e)*. This method (invoked by *isCellEditable(EventObject anEvent)* and *shouldSelectCell(EventObject anEvent)*) calculate the row being edited and sets the *editor* member to the coresponding editor (from the hash set).

Invocation of a method of the TableCellEditor (which is set as the CellEditor for the second column of the view) leads to invocation of the same method upon the selected editor.

7. Interaction between UML model and the property sheet

To acquire the information from the UML model to be displayed a hierarchy of adapter classes is created. This is similar with the metamodel class hierarchy. These classes have a method *getProperties(Element element)* for gathering properties and a method *propertyChanged(GProperty prop)* responsible for modification of properties of the current object (in our tool this responsibility is delegated a class who performs all operations upon the model). This method also made some check upon the new value of the attribute. It returns true if the update was successfully and false otherwise.

The steps performed when the properties of an element need to be shown are (notice the use, once more, of the MVC pattern):

- setting up the target element;
- invocation of the *updateView* method;
- creation of a new *PropertyModel* used by the JTreeTable to display the information;
- gathering information (after an adapter class of the UML element is created "P..."); interogation of the **model**;
- creation of nodes using the collected properties (root node is created with the target element as argument and is never displayed); creating the **view**;
- editors for each property are set up; setting the **controllers**.

8. Conclusions and further developments

The goal of this article was to present a solution for developing graphical components used to inspect the treats of objects which the user operates. Even if the example above is from the CASE tools world we tried to provide a solution which can be very easy adapted to any kind of application. In terms of MVC, only the model has to be updated to correspond to the needs of the application. (One has to subclass and full implement the *TreeTableModel* interface – see **Figure 2**).

A very important point about the property sheet is the graphical aspect. In this version this is quite simple. Its improvement will increase also the quality. These are some aspects that can be improved:

- Adding colours. It is useful to color attributes listed with different colors to easily differentiate between them (e.g. In the class properties, the color of inherited features may differ from the color of the ones defined by the class). Also, the attributes that cannot be modified may have a specific color to suggest they are read-only.

- Attaching icons to the leafs of the tree to provide additional information of the property (e.g. The features may have attached icons to illustrate their visibility).
- A pop-up menu context dependent and keyboard shortcuts to improve the editing proces of the model.

References

[1] Sun Java Tutorial, `http://java.sun.com/docs/books/tutorial/index.html`
[2] David M. Geary, *Graphic JAVA. Mastering the JFC*, vol II, Palo Alto, CA, 1999.
[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns*, Addison Wesley, Boston, MA, 1994.
[4] OMG Unified Modeling Language Specification, `http://www.omg.org`

Babeş-Bolyai University, Computer Science Research Laboratory, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania
E-mail address: `sorin@lci.cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLIX**, Number 1, 2004

APOLOGY ON PLAGIARISM PAPERS

THE EDITORS

Since the preceding issue has been send to print we have found out, and have been informed by more interested readers that the following papers are plagiates:

- D. Marcu, *The Chromatic Number of Triangle-Free Regular Graphs*, Studia Universitatis Babeş-Bolyai Series Informatica, 47 (1), 2002, p. 54–56.
- D. Marcu, *A Note on the Chromatic Number of a Graph*, Studia Universitatis Babeş-Bolyai Series Informatica, 47 (2), 2002, p. 105–106.
- D. Marcu, *A Note on the Chromatic and Independence Number of a Graph*, Studia Universitatis Babeş-Bolyai Series Informatica, 48 (2), 2003, p. 11–16.

According to practices currently in place, these papers have been reviewed, as always, by a panel of two experts. They have made all possible effort to ensure the scientific quality and accuracy of the papers submitted to the journal. However, we are not always able to verify the originality of every paper submitted, and, as usually, this rests with the responsibility of the author.

After a careful consideration, we have decided to retract the papers under scrutiny; the papers will be marked as such on the journal web page. As we have lost the confidence in Mr. Dănuţ Marcu, the author of these plagiates, we have decided to ban Mr. Marcu from publishing in our journal.

We are apologizing to the international scientific community for this situation. Despite this, we are ensuring our readers that we are continually working to ensure a high scientific quality for our journal. As such, they may continue to consider our journal as the journal of their choice.

The Editors

Faculty of Mathematics and Computer Science, Department of Computer Science, Babes-Bolyai University, 400084 Cluj-Napoca, Romania

E-mail address: studia-i@cs.ubbcluj.ro

Received by the editors: May 15, 2004.