Yugoslav Journal of Operations Research
5 (1995), Number 1, 65–77

AN OVERVIEW OF SOME TRUTH MAINTENANCE SYSTEMS

Mladen STANOJEVIĆ, Sanja VRANEŠ

Mihajlo Pupin Institute
P.O.Box 15, 11000 Belgrade, Yugoslavia

Dušan VELAŠEVIĆ

Faculty of Electrical Engineering
Bulevar revolucije 73, 11000 Belgrade, Yugoslavia

Abstract: Many treatments of formal and informal reasoning in mathematical logic and artificial intelligence have been shaped in large part by a seldom acknowledged view: the view that the process of reasoning is the process of deriving new knowledge from old, the process of discovering new truths contained in known truths. The basic problem with the conventional view of reasoning stems from the monotonicity of the reasoning process. The truth maintenance systems solve this problem. Doyle's *Truth Maintenance System* – TMS, de Kleer's *Assumption–based Truth Maintenance System* – ATMS, ART's *viewpoint* mechanism, and BEST's *context* mechanism – MEKON will be described in this paper. TMS and ATMS represent two different approaches in the truth maintenance, and some variations of these basic ideas are implemented in many other truth maintenance systems. ART's viewpoint mechanism and MEKON are similar to the ATMS, but they can solve some problems that neither TMS nor ATMS can solve.

Keywords: Expert systems, hypothetical reasoning, non–monotonic reasoning, time–state reasoning, truth maintenance.

1. INTRODUCTION

One of the most important aspects of intelligent behavior is the ability to reason about, and adapt to a changing environment, permanently reflecting perceived changes. Truth maintenance (sometimes also called belief revision) is an area of artificial intelligence concerned with the issues of revising sets of beliefs and "maintaining the truth in the system" when new information is found to contradict old

information. Typically, truth maintenance systems explore alternatives, make choices, explore the consequences of the choices and, if during this process a contradiction is detected, the truth maintenance system revises the knowledge base and gets rid of contradictions. There are many applications that can use and benefit from truth maintenance technique some of which are:

- **Planning systems**, that use truth maintenance to detect the source of problems and to prevent the generation of ill–formed plans;
- **Allocation tasks**, where the admissible or optimal allocations are determined;
- **Classification tasks**, where a member of a class is recognized upon a complete or incomplete, more or less accurate description;
- **Diagnostic systems**, that explore in parallel different potential diagnoses;
- **Decision making**, where different scenarios are made and examined to assess the consequences of the possible decisions and to choose the best one.

It is interesting to notice the ease with which we solve the apparent contradictions involving our commonsense beliefs about the world. We routinely make assumptions about the causes of events, reactions, attitude, or about properties and permanence of the objects, yet we easily make the necessary corrections in favor of new evidences. Thus, the set of our commonsense beliefs changes non–monotonically.

Our beliefs of what is current also change non–monotonically. In one moment of time we can believe that one assumption is true, while in the other, we can reject it based on some new evidences. We usually make our decisions based on what we currently believe, so we must continually update our current set of beliefs. The problem of describing and performing this updating efficiently is sometimes called the *frame problem*.

The third problem with the conventional view actually subsumes the problem of commonsense reasoning and the frame problem. The problem of control is the problem of deciding what to do next. To make this choice blindly isn't obviously the best possible strategy.

Four different ways of belief revision, Doyle's *Truth Maintenance System* – TMS [4], de Kleer's *Assumption–based Truth Maintenance System* – ATMS [1], [2], [3], ART's *viewpoint* mechanism [5], and BEST's *context* mechanism – MEKON will be described and compared in this paper.

2. TRUTH MAINTENANCE SYSTEM – TMS

The field of truth maintenance is usually recognized to have been initiated by Doyle, McAllester, McDermott, de Kleer, and Martins. Doyle's Truth Maintenance System – TMS [4] relies on *justifications* of beliefs in its work. It manipulates two data structures: *nodes*, which represent beliefs, and *justifications* – the reasons for the TMS to believe or disbelieve a certain proposition. As its fundamental actions, the TMS can:

- create a new node, to which the problem solver using the TMS can attach the statement of a belief. The manipulation of these statements is left to the problem solver using the TMS. Nodes are referenced using indexes (N–1, N–2, etc.);
- add (or retract) a new justification for a node, as a result of application of a rule or procedure (thus providing non–monotonic reasoning). Rules and procedures also have the TMS nodes, which they include in justifications they create;
- mark a node as a contradiction, to represent the inconsistency of any set of beliefs.

A node contains a proposition and several justifications representing different reasons for believing the node (e.g., "N–1 proposition1 justification1"). The node is believed if and only if at least one of its justifications is valid. The TMS uses two types of justifications: *support–list* (SL) justifications, and *conditional–proof* (CP) justifications. A justification is a *well–founded supporting justification* if all of its arguments (nodes) are non–circular.

The SL justifications contain two lists of nodes: *inlists* and *outlists*. They have the following form: (SL <*inlist*> <*outlist*>). The *inlist* and *outlist* are created by the problem solver as the result of rule firing or procedure execution. Nodes used by that rule or procedure comprise the content of the *inlist* and *outlist*. The proposition supported by an SL justification is believed if and only if every proposition in its *inlist* is believed, and every proposition in its *outlist* is disbelieved. Based on SL justifications, there are two types of specific propositions in TMS. *Premises* are propositions whose current SL justifications have empty *inlists* and *outlists* (i.e., they are always believed), while *assumptions* are propositions whose current SL justifications have nonempty *outlists*. The meaning is the following: until there is no evidence to the contrary (all nodes from the *outlist* are out), and all the nodes representing the reasons for believing it are in (all nodes from the in *inlist* are in), the assumption is in. The assumptions have non–monotonic justifications.

The CP justifications have different structure from SL justifications. They contain a consequent, a list of *inhypotheses*, and a list of *outhypotheses*: (CP <consequent> <inhypotheses> <outhypotheses>). A CP justification is valid if the corresponding consequent node is in whenever each node of the *inhypotheses* is in, and each node of the *outhypotheses* is out. Each time the TMS finds a CP–justification valid it computes an equivalent SL justification.

The algorithm for the addition of justification is presented in Figure 1. The algorithm for the retraction of justification is similar.

The TMS solves the problems that stem from monotonicity. The set of beliefs can change non–monotonically, thus solving the commonsense reasoning problem and the frame problem, while the control problem must be resolved by the problem solver.

```
add_justification(in:node, in:justification
    add a new justification;
    update the beliefs of the affected nodes;
    if there are CP justifications then
            process CP justifications and
            evaluate equivalent SL
            justifications;
    if there are contradictions then
            call dependency-directed
            backtracking;
    signal changes to the problem solver;
```

Figure 1. Algorithm for the addition of justification in TMS

3. ASSUMPTION-BASED TRUTH MAINTENANCE SYSTEM

Following Doyle, de Kleer developed an Assumption-based Truth Maintenance System - ATMS [1], [2], [3], which is, for many tasks, more efficient than the TMS. De Kleer mentioned some of the TMS limitations that are eliminated in the ATMS:

- even when a problem has more than one solution, the TMS algorithms only allow one solution to be considered at a time. This makes it extremely difficult to compare two equally plausible solutions, and to find the best solution;
- the TMS forces overzealous contradiction avoidance. If A and B are contradictory, the TMS guarantees that either A or B will be worked on, but not both. However, only inferences dependent on both A and B should be avoided, while the inferences dependent on only A or B should be drawn;
- switching states in the TMS is difficult. An assumption can be changed only by introducing a contradiction, but once added it cannot be removed so the knowledge state of the problem solver is irreconcilably altered;
- an assumption in the TMS is any node whose current supporting justification depends on some node being out. The set of assumptions changes during the problem solving and this causes problems to the problem solver which frequently consults the assumptions and justifications for data;
- the TMS algorithms spend a surprising amount of resources finding a solution that satisfies all the justifications;
- the process of determining which retracted, previously derived, data can be reasserted is called unouting. Unfortunately, unless great care is taken at the problem-solver-TMS interface, some previously discovered data will be rederived.

In the ATMS, a *node* corresponds to a problem-solver datum, while a *justification* describes how a node is derivable from other nodes. A justification has three parts: the

node being justified, called the *consequent*; a list of nodes, called the *antecedents*, and the problem solver's description of the justification, called the *informant*. An ATMS *environment* is a set of assumptions. Logically, an environment is a conjunction of assumptions. A node holds in an environment if it can be derived from the set of assumptions and the set of justifications. An environment is inconsistent if false is derivable propositionally. The set formed by assumptions and nodes derivable from them comprises an ATMS *context*.

Every node is associated with a set of environments: this set is the node's *label*. The label describes the assumptions the datum ultimately depends on and unlike justifications, is constructed by the ATMS itself. While a justification describes how the datum is derived from immediately preceding antecedents, a label environment describes how the datum ultimately depends on assumptions. The basic task of the ATMS is to guarantee that each label of each node is consistent, sound, complete and minimal. A label for a node is: *consistent* if all its environments are consistent, *sound* if the node is derivable from each environment of the label, *complete* if every consistent environment E in which the node is derivable, is a superset of some environment E' of nodes label, and *minimal* if there are not two environments in the node's label such that one is the superset of the other.

There are four types of nodes in the ATMS: *premises*, *assumptions*, *assumed nodes*, and *derived nodes*. A premise has a justification with no antecedents, i.e., it holds universally. The node,

$$< \text{p}, \{\{\}\}, \{0\} >$$

represents the premise p. An assumption is a node whose label contains a singleton environment mentioning itself. The node,

$$< \text{A}, \{\{\text{A}\}\}, \{(\text{A})\} >,$$

represents the assumption A. An assumed node is neither a premise nor an assumption, and has a justification mentioning an assumption. The assumed datum a which holds under assumption A is represented by

$$< \text{a}, \{\{\text{A}\}\}, \{(\text{A})\} >.$$

All other nodes are derived. The derived node,

$$< \text{w=1}, \{\{\text{A, B}\}, \{\text{C}\}, \{\text{E}\}\}, \{(\text{b}), (\text{c, d})\} >$$

represents the fact that $w = 1$ is derived from either the node b or nodes c and d. In addition, the node holds in environments {A, B}, {C} and {E}.

The three basic ATMS actions are creating an ATMS node for a problem solver's datum, creating an assumption, and adding a justification to a node. The algorithms ensure that, after every primitive operation, every label of every node is consistent, sound, complete and minimal.

The basic algorithm in ATMS is described in Figure 2.

In the ATMS, a *consumer* is a rule which does some processing on the datum. Consumers are attached to nodes, run when appropriate and only once for a given node. The order of consumer execution has no effect on the final problem–solver state, but, it has a significant effect on efficiency. In the ATMS an environment is scheduled

first, and then the consumers dependent on the scheduled environment are executed. The problem of control is left to the scheduler and has little to do with the internals of the ATMS.

```
add_justification(in:node, in:justification)
    compute a new label from the
    justification;
    remove inconsistent and subsumed
    environments;
    if the new label is not the same as the
      old label then
            if the node represents contradiction
              then
                    add each environment of the
                    label to a nogood database;
                    remove all inconsistent
                    environments from every
                    node label;
            else
                    update the labels of all the
                    consequent nodes
            end if;
    end if;
```

Figure 2. The basic algorithm in ATMS

4. ART'S VIEWPOINT MECHANISM

Automated Reasoning Tool – ART is a powerful tool for expert system development [5]. The truth maintenance system similar to the ATMS (but not the same), is implemented in ART through the use of the *viewpoint* mechanism. The *viewpoint* mechanism in ART is strongly influenced by the fact that ART is a tool for expert system development.

Expert systems find their use in solving the problems for which the applicable algorithms don't exist, or if those algorithms exist, their use is too expensive. Typical problems suitable for the expert systems are the problems that can be solved by searching the solution space. The problems that neither TMS nor ATMS can cope with, are the problems where every point in the solution space is defined by a fixed number of relevant variables. Every of these variables can take many values, but for one point every variable has exactly one value.

The difficulties in solving this kind of problem in TMS arise from the fact that only one datum representing the variable's value can be *in* at a time. All the data representing other variable values must be *out*. If there are only two variable values *A*

and *B*, then the SL justification for the datum *A* would be (SL () (B)), while the SL justification for the datum *B* would be (SL () (A)). Those SL justifications are not well founded and thus not permitted in the TMS.

The ATMS cannot be used to solve this kind of problem, because it is not able to generate a new assumption using derived nodes. The ATMS can solve these problems only if all the possible assumptions (dependent on the problem state) are known in advance. The number of the possible assumptions is very large even for a small solution space. In order to find all the possible assumptions it is necessary to examine each point of the solution space, and thus to solve the problem (in which case we don't need the ATMS at all).

In the ATMS a datum once asserted cannot be retracted later. This is not the case in ART. This has the advantage in that it allows negated assumptions, but doesn't allow disjunctions of *extents*. Thus, multiple justifications can cause problems for ART. If a second justification is found for a datum, and this second justification holds in an environment which is not the same as, a subset of, or a superset of the environment supplied by the first justification, then a new node with the same datum must be created to accommodate the new environment and justification.

A node in ART contains a datum, and an extent which corresponds to a label in ATMS terminology. Unlike the TMS and ATMS, ART's *viewpoint* mechanism doesn't use justifications in the creation of labels. The *viewpoint* mechanism creates explicit contexts called *viewpoints*, and a datum is always asserted in a current *viewpoint*.

An extent describes in which *viewpoints* a datum holds. In ART there exists one *meta viewpoint* level (containing a *meta viewpoint*), and an arbitrary number of user's *viewpoint* levels, Figure 3). Facts asserted into the *meta viewpoint* are premises (in TMS terminology), because they are visible from all other *viewpoints*. Their extents are *meta*. When a fact is retracted from the *meta viewpoint*, then the whole node representing it is retracted. The *meta viewpoint* level is always present, while the user's *viewpoint* levels must be created explicitly.

User's *viewpoint* levels are hierarchically ordered. The name of a *viewpoint* from the first user's *viewpoint* level consists of the *viewpoint* level name and the number of *viewpoint*. The name of a *viewpoint* from the second *viewpoint* level is composite and consists of the name of the current *viewpoint* from the first level and a name that contains the name of the second level and the number of the *viewpoint*. The names from other *viewpoint* levels are defined in a similar manner. The inheritance is defined between *viewpoints* from the same *viewpoint* level. The descendant *viewpoints* inherit the data from the ancestor *viewpoints*. The data on the *meta viewpoint* level are visible from every user's *viewpoint* level, while the data from a *viewpoint* in an outer user's level are visible in all *viewpoints* defined in inner levels. When a datum is asserted into a *viewpoint* at the user's *viewpoint* level, the corresponding extent will contain the name of that *viewpoint*. If a datum is retracted from a *viewpoint* on the user's level, then it is *shadowed* in that *viewpoint*, thus becoming invisible in that *viewpoint*. The extent of this datum will contain the name of the *viewpoint* where the datum was asserted, and the names of the *viewpoints* where it was retracted. If the same datum is

asserted in an offspring of the *viewpoint* where it was retracted, then a new node must be created for that datum.

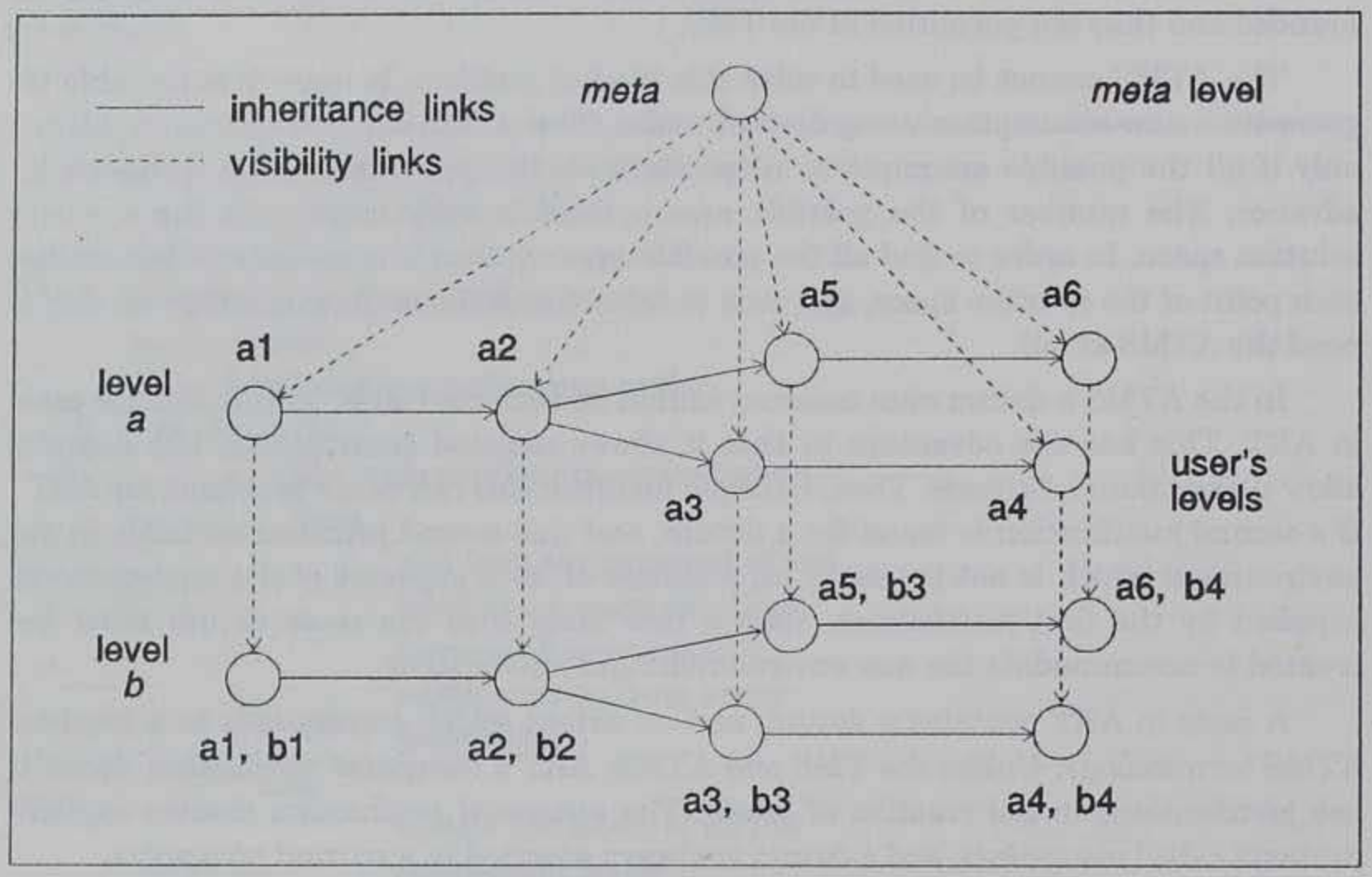

Figure 3. Viewpoint levels in ART

In ART the declarative knowledge is represented using *hypotheses*, *facts*, and *schemata* (frame–like objects implemented using facts). Hypotheses correspond to assumptions in ATMS terminology, while facts correspond to premises, assumed and derived nodes. Hypotheses are used in hypothetical reasoning, cannot be retracted and must be unique. Hypothetical reasoning subsumes the problems that can be solved using ATMS, and the problems that change the state using some assumptions. Facts are used in the time–state and hypothetical reasoning. Time–state reasoning is used to reason about the set of facts that change in time. Facts are used to represent the state variables.

The basic algorithms of the ART' s *viewpoint* mechanism are simple. When a new datum is asserted, a new node is created, if this datum is not inherited from the ancestor *viewpoint*. When a datum is retracted from the current *viewpoint*, then the extent of the corresponding node is *shadowed*.

ART' s rule language provides a rich set of commands for the use of *viewpoint* mechanism. It is possible to create new *viewpoints*, to change their contents, to *poison* them (to delete a *viewpoint* if it contains a contradiction), or to merge them. The set of commands for the change of the current *viewpoint* level or the change of the current *viewpoint* is also available.

The problem of control in ART is solved using the agenda mechanism in rule scheduling. The depth–first search strategy with the use of priorities is implemented

as the control strategy in ART. MEKON has been developed in an attempt to overcome this rigidity in control and other drawbacks of ART, TMS and ATMS.

5. BEST'S CONTEXT MECHANISM – MEKON

Blackboard-based Expert System Toolkit – BEST is a complex multiparadigm tool for the expert systems development [8] that uses the blackboard architecture. BEST's context mechanism – MEKON is in its basic ideas similar to the ATMS, while the rule language which enables the use of MEKON is similar to ART's rule language. The implementation of MEKON was influenced by its application in expert systems.

MEKON provides belief revision, contradiction handling and non-monotonicity handling. Belief revision must be performed whenever a state transition is performed to reflect the new values of state variables. Contradiction handling is used when an inconsistent problem state is discovered. This problem state will be removed from the decision tree and the examination of the unproductive branch will be stopped. Contradiction handling improves the efficiency of the overall system by constraining the search space that must be examined to find problem solutions. Non-monotonicity handling allows solution of search problems with incomplete information. Without MEKON, a user would have to implement these features from scratch each time he programs an application to solve a search problem. Hence, the time needed to implement such applications can be significantly reduced if MEKON is used.

Using MEKON all kinds of problems that can be solved by ART, can be solved by BEST too. The differences between *viewpoint* mechanism in ART and MEKON are in the implementation of the truth maintenance system, control, and the rule language. The rule language in BEST that facilitates the work with MEKON is more comprehensive and easier to use than ART's rule language.

MEKON is an AI technique for the simultaneous exploration of alternative hypotheses leading to possible problem solutions. It might be of enormous help when there is uncertainty in determining a solution to a problem. The essence of the technique is that a separate context is created for each point in the solution space examined during the problem solving. A context is a label for a set of *hypotheses*, *facts* and *concepts* (frame-like objects) [6] that comprise the content of this context. Hypotheses correspond to assumptions in the AMTS terminology, while the slot values of concepts and facts correspond to the values of state variables. MEKON is also used to enhance the efficiency of problem solving by sharing information across a solution search space.

The use of MEKON allows for the maintenance of models of multiple hypothetical situations. States of the problem represented by contexts can be analyzed in detail and compared using rules. When an undesirable context is generated, it is detected by a special kind of rules (*constraint* rules) and a bad search path is poisoned. The exploration of the search space is then continued using rules that generate other states.

A node in MEKON contains a datum and a context to which this datum belongs. This solution requires the existence of one node for each instance of a datum in a context. There can be many nodes in the system containing the same datum defined in different contexts at the same time. This redundancy is utilized for efficiency reasons (the classical time–space tradeoff).

A problem to be solved by BEST can be decomposed into many subproblems which will be solved using Domain Knowledge Sources – DKSs. The DKSs are the basic computation agents that can be programmed using the most appropriate programming paradigm for the given subproblem. The DKSs communicate with each other using Global Domain Blackboard – GDBB. In order to provide locality of data, every DKS can have its local blackboard – Local Domain Blackboard (LDBB).

Each blackboard (global or local) contains two context levels – *meta* and user's level (Figure 4). The *meta* context level consists of a single *meta* context, while the arbitrary number of contexts comprises the user's context level. A user level context represents a set of facts and concepts said to be true in that context. There is also a set of facts and concepts defined at *meta* level that are visible in every context at the user's level, thus behaving as premises (in TMS terminology). Contexts represent a collection of changes over time or hypothetical situations. Contexts on the user's level inherit all data from their ancestor contexts. If desired, the inherited data can be deleted selectively, and new data can be asserted. Contexts comprise the structure of each blackboard.

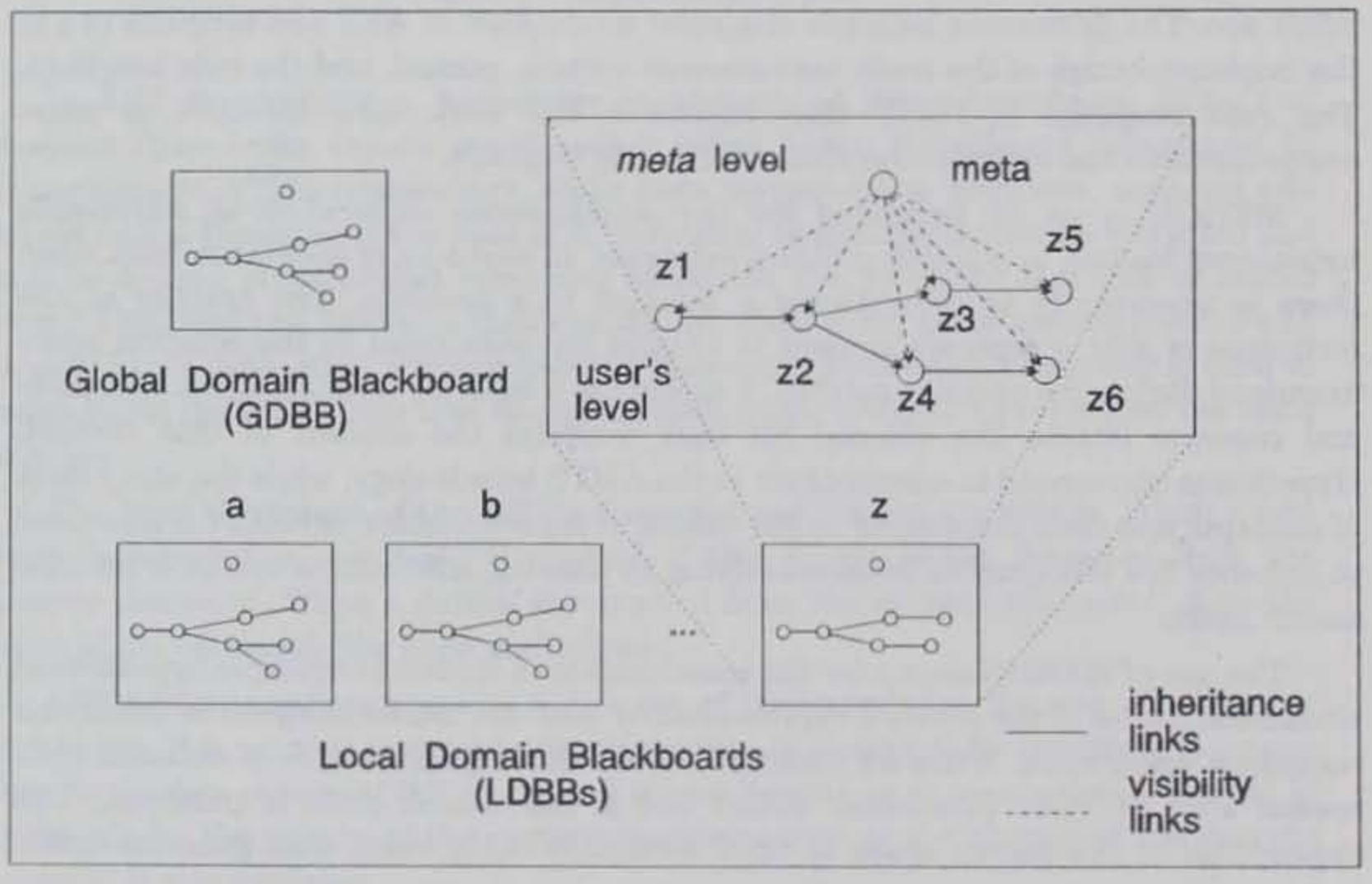

Figure 4. Blackboard layout in BEST

Every blackboard corresponds to one domain and has a unique name. The name of a domain that corresponds to the GDBB is *global*, while the names of domains corresponding to the LDBBs are defined by a user. Those names are used for making the names of contexts on the user's levels. The domain name combined with the number of created contexts increased by one makes the name of a new context. The existence of only two context levels on each blackboard simplifies a lot the use of MEKON. There is only one visibility rule for the data: all data defined in the *meta* context on one blackboard are visible in all contexts at the user's level on the same blackboard.

The data within each context are internally consistent, but data across the contexts may be contradictory. That is precisely the point of creating many contexts – to explore the implications of contradictory data. However, contexts that violate constraints established by the problem solver through the *constraint* rules are deleted (poisoned), reducing the number of contexts to be searched and limiting the solution space.

A datum is always asserted into the current context on the GDBB or the current LDBB. The same holds for the retraction of a datum. When a datum is asserted in the current context, it is checked first if the corresponding node exists. If such a node exists, there is nothing to do, otherwise a new node is created. When a fact is retracted from the current context, the existence of the corresponding node is checked first. If that node is defined, it is retracted, but if it isn't, there is nothing to do.

The basic algorithms in MEKON are simple and efficient. When a datum is asserted, only a new node should be created. If a datum is retracted, then the corresponding node will be deleted. During the creation of a new context, all the nodes that belong to the ancestor context are copied, thus becoming available in the created context. MEKON maintains the structure of contexts by updating the lists of ancestor, and descendant contexts for each context, as well as the context depth in this structure.

BEST's rule language allows a user to exploit the full power of MEKON. It is possible to create, poison, change (by asserting, retracting, or modifying data), or merge contexts. The change of the current context, current context level (*meta* or user), as well as switching between GDBB and current LDBB are also permitted.

BEST user interface facilitates a graphical representation of the context content at any time. Each context (network node) can be moused to show the hypotheses, facts, and concepts associated with the context. It is also possible to add or delete interactively a fact or concept, what is of enormous help in "what if" analyses and problem reformulating. The context can be analyzed and problem solver rerun with different data from that point further, rather than reinitializing and rerunning the entire problem.

The control in BEST is hierarchically organized. The higher level of control is responsible for the DKS scheduling and activation, while the lower level determines the rule firing within the DKS execution. The higher level of control is implemented using Domain Knowledge Source Activation Rules (DKSARs), depth–first strategy with priorities, Knowledge Source Agenda (KSA), criterion (if one is defined) and

meta–rules. To each DKS corresponds one DKSAR whose if–part contains all the preconditions necessary for the DKS activation, while then–part initiates the DKS execution. If many DKSs are solving the same subproblem, it is possible to define a criterion to determine which of them will actually be executed. The criterion can be the DKS' efficiency, value, reliability or a heuristic function that combines those criteria. The *meta*–rules can change the sequence of DKSs execution using the data on the GDBB and the KSA. The lower level control is very flexible. Using *set–control* rules [7] different search strategies with or without priorities (depth–first, breadth–first, A*, hill–climbing, branch–and–bound, beam–search, best–first, etc.) can be applied. *Meta*–rules can change the order of rule firing using the data on the LDBB, GDBB, and the rule agenda.

Problem solving is typically an exploratory, incremental process. That is, one starts with a set of beliefs about the world and then considers alternative choices that modify these beliefs. BEST allows us to reflect the structure of the solution space in the structure of the context network. One can model alternatives or changes to a particular context by creating a descendant context. Different search strategies and *meta*–rules are implemented in BEST to explore efficiently the alternative pathways to solutions.

6. CONCLUSIONS

In this paper an overview of some truth maintenance systems is presented. Doyle' s TMS and de Kleer' s ATMS are the representatives of two different approaches in resolving the truth maintenance problems. The TMS is based on the use of justifications, while the ATMS relies on the use of assumptions. The *viewpoint* mechanism in ART, and MEKON are variations of the ATMS, but contain some differences which stem from the tuning of their truth maintenance systems to enable the solving of some problems that can arise in expert systems' application. MEKON is an ATMS–like truth maintenance system applied to the blackboard architecture. The most important details of these four truth maintenance systems are presented in this paper with the emphasis on how the problems of commonsense reasoning, frame problem, and the problem of control are resolved in these systems. To some extent these four systems are compared with each other by analyzing their advantages and disadvantages.

REFERENCES

[1] de Kleer,J., "An Assumption–based Truth Maintenance System", *Artificial Intelligence* 28 (1986) 127–162.

[2] de Kleer,J., "Extending the ATMS", *Artificial Intelligence* 28 (1986) 163–196.

[3] de Kleer,J., "Problem Solving with the ATMS", *Artificial Intelligence* 28 (1986) 197–224.

[4] Doyle,J., "A Truth Maintenance System", *Artificial Intelligence* 12 (1979) 231–272.

[5] *ART – Automated Reasoning Tool User's Manual*, version 3.0, Inference Corporation, Los Angeles, CA, 1987.

[6] Vraneš,S., and Stanojević,M., "Prolog/Rex – A Way to Extend Prolog for Better Knowledge Representation", *IEEE Transactions on Knowledge and Data Engineering* 6/1 (1991) 22–37.

[7] Subašić,P., Stanojević,M., Stevanović,V., and Vraneš,S., "Adaptive Control in Rule Based Systems", in: A.Moudni, P.Borne, S.Tzafestas (eds.) *IMACS Symposium Modeling and Control of Technological Systems*, Lille, France, Gerfidn, May 1991, 764–769.

[8] Vraneš,S., Stanojević,M., Lučin,M., Stevanović,V., and Subašić,P., "A Blackboard Framework on Top of Prolog", *Expert Systems with Applications* 7/1 (1991) 109–130.