A Logic-Based Approach To Automated System Management

Along Lin
Internet Business Management
HP Laboratories Bristol
HPL-98-19
February, 1998

E-mail: alin@hplb.hpl.hp.com

diagnosis,
model-based-reasoning,
planning,
logic programming

In this paper, a logic-based approach to automated system management is described. In particular, we focus on model-based fault diagnosis and fault correction based on planning.

A Logic-Based Approach to Automated System Management

Along Lin

Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
Bristol BS12 6QZ, U.K.
Email: alin@hplb.hpl.hp.com

Abstract

In this paper, a logic-based approach to automated system management is described. In particular, we focus on model-based fault diagnosis and fault correction based on planning.
Keywords: Model-Based Diagnosis, Planning and Logic Programming.

1 Introduction

The explosive growth of the Internet has made management of networks and systems much more complicated. Although there may be various kinds of functions involved in network and system management, the management tasks such as configuration, performance monitoring, fault diagnosis and repairing faults are commonly associated with AI techniques because they are knowledge intensive. Knowledge-based systems are needed to significantly simplify management tasks. We believe an automated management system will play a very important role in providing customers with high quality services over networks.

Existing system management tools solve configuration and repairing problems by executing some pre-written programs or scripts, which are inflexible and have to be re-coded to adapt to different platforms. If the management is policy-based, system managers have to modify existing executables to enforce policies. Furthermore, it is impossible to generate all of the possible executables in advance since there are too many cases to be considered. In order to tackle this challenging problem, an innovative way of generating scripts or sequence of primitive actions automatically must be utilised. A lot of research has been done on planning techniques with some exciting results [1,3,4,6,13,14,16,17]. In this paper, a heuristic Partial Order Planner with Universal quantification's and Conditional effects (UCPOP) is described.

Model-Based Reasoning (MBR) [2, 5, 7-12,15] has been widely accepted as the principal diagnostic technique in several domains. However, little emphasis has been put on its

application to network and system management. In MBR, it is assumed that the structure of a system is known and we can use that knowledge to reason about its normal behaviour. Because it can diagnose faults which have not been pre-determined, it is a more promising technique than other knowledge-based approaches such as Rule-Based Reasoning, Case-Based Reasoning, Bayesian Networks, and Neural Networks. A Model-Based diagnostic system consists of a fault detection mechanism and a diagnostic engine. Some autonomous systems can also include a recovery mechanism to rectify identified faults.

In the following, first an automated system management prototype is described, then the idea of model-based fault diagnosis is presented. In section 4, a heuristic partial order planning algorithm is proposed. Finally, some issues concerning the application of AI technologies to practical problems are addressed.

2 An Automated System Management Prototype

FLIPPER is a prototype automated management system which manages devices such as workstations, Unix, Netware servers, PC's, printers, routers. It is based on Logic Programming technology and like a former prototype Dolphin, also developed by Hewlett-Packard Laboratories, Bristol. FLIPPER has a typed logic programming language. It supports type definition and an inheritance mechanism that allows the methods in a super object type to be overridden by those defined in subtypes. However, it does not allow a subtype to add new attributes in addition to those present in super types. It consists of a model compiler, an inference engine, and a diagnostic engine. Different from most existing object-oriented logic programming languages, it is not implemented by adding an extra layer on top of some logic programming system, which doesn't support fault diagnosis, monitoring, and fault correction.

An object type in FLIPPER is described in terms of its attributes and behaviors. The behaviors of an object are defined by querying rules, access methods and actions. A relationship among several objects is defined by a set of rules which have the same relation signature. Each rule is described as a typed Horn Clause. Access methods link to the real world to obtain or check the information about managed objects by invoking external functions. Although there is a difference in the way they are defined, rules and access methods are dealt with consistently by the underlying inference engine. Actions are used to change the states of the managed objects. The state of an object is represented by its attributes.

FLIPPER manages a system in a goal-directed way. Administrators use goals to express their desires of management tasks. There are two special types of goals. Monitored goals are always watched and used to trigger the diagnostic engine or generate an event to signal some state change in the managed system to a manager. Resident goals

need to be maintained true, which involves monitoring and fixing. As a resident goal becomes false, a sequence of executable primitive actions will be executed automatically, then the system will be in a consistent state again. While an inference engine is used to prove goals, a diagnostic engine is provided to diagnose faults. Repairing in FLIPPER is pre-writing scripts for various purposes, however, we expect to utilize a planner to automatically generate necessary primitive actions to rectify diagnosed faults.

3 A Goal-Directed Approach to Fault Diagnosis

Guarded Horn Clauses (GHCs) are used to express model rules. A GHC rule R is expressed as: H IF G | B, where H , G , and B are the head, guard and body respectively. Goals are divided into diagnosable ones, which model writers think are relevant to the possible faults, and non-diagnosable ones. For a fault diagnosis, G does not contribute to any faults in a system, it is just used to commit to a rule R. Although backtracking is allowed within G, B is deterministic. A guarding goal must not contain any diagnosable goals.

The failure tree of a goal in an AND/OR tree can be defined as follows:

- for an AND node, which has sub-goals in the body of a committed rule as its children, any one of them is in the failure tree;
- for an OR node, which has all potential committed rules as its children, its children are in the failure tree;
- for a leaf node, if it fails, it is in the failure tree;
- only those nodes mentioned above are in the tree;

Based on the definition of the failure tree, the diagnosis of a failed goal is actually to find a solution tree in which the root goal has the *false* truth-value. However, the solution in an AND/OR tree can have a large amount of irrelevant information. In [2], a method was proposed to tackle this issue. The search proceeds by expanding an unknown node and propagating the truth-values of atomic access methods or facts in a bottom-up way alternatively. Meanwhile, for those nodes which have *true/false* truth-values, all of their children can be ignored for single cause or fault. The diagnosis of the failed goal G consists of all the leaf nodes in its solution tree. Obviously, in order to diagnose multiple faults, we should continue to search for other faults after collecting a fault into a returning solution list. However, this is restricted by the dependence of the fault on other sub-goals in a rule's body. If the fault sub-goal is expected to produce some values which are inputs to other goals, we cannot diagnose the faults which are the leaf nodes of those goals.

Consider an example of a user securely printing some information contained at a remote site to a printer.

```
MODEL PRINTING
IMPORT Resource Site User Printer
OBJECT SecurityClass ISA Integer
RULES
[User u] location [Site s1] print [Resource r] at [Site s2] with [Printer p]
(+ + + + ?) (+ + + + +) IF  /* p can be bound/unbound, the others must be bound */
        [r] at [s2] securityClass [SecurityClass sc],
        [u] userAt [s1] securityClass [SecurityClass sc1],
        [p] printerAt [s1] securityClass [SecurityClass sc2],
        [sc] notLessThan [sc1],
        [sc2] notLessThan [sc1],
        [sc] notLessThan [sc2]
        |    /* diagnosable goals can appear after committing to this rule */
        [s1] secureCommunication [s2],
        [u] userAt [s1] authorizedToAccess [r] at [s2],
        [u] userAt [s1] authenticatedAndConnectedTo [s2],
        [p] location [s1] isInWorkingCondition.
```

If [alin] location [hplb] print [article] at [ieee] with [Printer p] fails, where hplb, article ieee and alin have been bound to some particular objects, there will be several possible causes: the connection between two sites is insecure, the user is not allowed to access the specified resource, or there is no available printer to print it out which meets security restrictions and is in a working condition. Because of the independence of the sub-goals, which are diagnosable, all of the faults could be found. The definition of [] printerAt [] securityClass [] is as follows:

```
[Printer p] printerAt [Site site] securityClass [10] IF
        [site] name ["hplb"], [p] name ["elm"].
[Printer p] printerAt [Site site] securityClass [8] IF
        [site] name ["hplb"], [p] name ["oak"].
[Printer p] printerAt [Site site] securityClass [3] IF
        [site] name ["hplb"], [p] name ["amazon"].
```

4 A Heuristic Partial Order Planning Algorithm

4.1 The Representation of Actions

An action describes a precondition, which must be true before it can be invoked, and a post-condition, which will be true after it is executed. Its syntax is defined as follows:

```
<action>     ::= <relation> 'PRE' <pre> 'POST' <post> '.'
<goal>       ::= <relation>|'¬'<relationl>|<goal>{'&'<goal>}*|<goal>{'|'<goal>}*
<non-code>   ::= <var1>'≠'<var2>{'&'<var1>'≠'<var2>}*
<pre>        ::= <goal>['&'<non-code>]|∀x∈T'('<pre>')'
```

<effect> ::= <relation>|'¬'<relation>|<effect>{'&'<effect>}*
<conEffect> ::= <goal>'→'<effect>
<post> ::= <effect>|<conEffect>|$\forall x_{\in T}$'('<post>')'|<post>{'&'<post>}*
In <conEffect>, <goal> refers to the world before the action is executed while <effect> refers to the world after execution. In our language, $\forall_{x \in t} Q(x)$ can be expressed as:

[Type x] forAll Q(x)

For example, [Employee e] forAll ([e] authorisedToUse [Printer p]) means every employee is authorized to use printer p. $\forall_{x \in t} Q(x)$ can be replaced by its universal base $Q(c_1)\&\ldots\&Q(c_s)$ if $T = \{c_1,\ldots, c_s\}$, providing the world is static and making the Closed World Assumption (CWA). In addition, the variables in goals are implicitly existentially quantified. Since UCPOP[13,17] has been proved both sound and complete, we modify it to adapt to our action representation and improve on it by adopting a heuristic search technique based on the evaluation functions for increasing its efficiency. It is assumed that the initial state contains no variables and all the variables in the effects of an action must appear in its preconditions.

4.2 Action Examples

ACTIONS
[VolGroup vg] rmVolGroup // action 1: remove a volume group vg on a machine m if
PRE [vg] vgMachine [Machine m] &
[vg] onlyOnePV & // there is only one physical volume on volume group vg
[vg] noneLV // there are no logical volumes on volume group vg
POST ~[vg] vgMachine [m] &
~[pv] belongsTo [vg].
[VolGroup vg] rmPhVol [PhysVol pv]
// action 2: remove a physical volume pv from a volume group vg if
PRE [pv] belongs [vg] & // physical volume p belongs to volume group vg
[PhysVol p] belongs [vg] & // there is another physical volume p(≠ pv) in vg
p ≠ pv &
[LogVol lv] forAll (~[pv] logPart [lv]) // pv doesn't belong to any logical volume
POST ~[pv] belongs [vg].
[LogVol lv] rmLogVol // action 3: remove a logical volume lv from volume group vg
PRE [lv] on [VolGroup vg] &
[Swap sw] forAll (~[sw] swapSpace [lv]) // there is no swap space in lv
POST ~[lv] on [vg] &
[PhysVol pv] forAll (([pv] logPart [lv]) → (~[pv] logPart [lv])).

RULES
[VolGroup vg] onlyOnePV IF
[PhysVol pv] belongsTo [vg] & [PhysVol p] forAll (~[p] belongsTo [vg] | pv).
[VolGroup vg] noneLV IF [LogVol lv] forAll (~[lv] on [vg]).

4.3 A Heuristic Partial Order Planning Algorithm

In the heuristic planning algorithm below, we use the following functions and notations:
pre(A) returns pre-conditions of an action A;
post(A) returns post-conditions of an action A;
$g(G)$ returns 1 if goal G can be achieved by an access method; Otherwise, it returns 0;
link: *actions* × *literals* × *actions*, $<A_p,G,A_c>$ means that G is both an effect of A_p and a precondition of A_c, *literals* is the set of all relations and negated relations;
condeffect: *conEffects* → *seqs*, where *conEffects* is the set of conditional effects and *seqs* is the set of consequent of conditional effects;
conditional(o_i,e,Q) is *true* if an action o_i has a conditional effect e and the goal $Q \in condeffect(e)$; Otherwise, it is *false*;
condgoal: *conEffects* → *antes*, *antes* is the set of all antecedents of conditional effects;
noncode: *actions* → *nons*, where *nons* is the set of all non-codesignations;
candidates: *goals* → *action_set*, mapping a goal g to the set of actions each of which has an effect Q or a conditional effect e and *condeffect*(e) contains Q such that Q and g have same relation signature, *action_set* is the power set of *actions*;
achieve(o_i,G,Q,θ,δ) is *true* if $o_i \in candidates(G)$ and $(G \bullet \theta)\bullet\delta == (Q \bullet \theta)\bullet\delta$, Otherwise, it is *false*;

Two heuristic evaluation functions are used to choose a goal to resolve with and an action to achieve it based on least-commitment principle.

$f(G) = |candidates(G)| + g(G) - 1$, where G is a literal;
$f(\neg G) = f(G')$, where, $G' \equiv (\neg G)$;
$f(g_1 \& g_2 \& \ldots \& g_n) = \sum_{i=1}^{n} f(g_i)$;
$f(g_1 | g_2 | \ldots | g_n) = min\{f(g_i)\},\ i = 1,\ldots,n$
$f(\forall x \in T Q(x)) = |T| \times f(Q(x))$;

The h-value of an action is defined as: $h(op) = f(pre(op))$.
By compiling models, we can calculate the values of those functions, for each relation R, *candidates*(R) can be sorted based on the h-values, thus reducing the cost caused by the heuristic algorithm greatly.

A_0 is an action which has no preconditions and its effects specify what is true in the initial state of a planning problem. A_∞ is another special action which has no effects, and has the goal of the planning problem as its precondition. Before the algorithm is called, let $A = \{A_0, A_\infty\}$; $O = \{A_0 < A_\infty\}$; $L = \phi$; If the goals we are achieving are $\{g_1, g_2, \ldots, g_n\}$, assign $\{<g_1,A_\infty>, <g_2,A_\infty>, \ldots, <g_n,A_\infty>\}$ to *Goals*. Initially, $\theta = \phi$.

Algorithm 1: **planning**($<A,O,L,\theta>$, *Goals*)

1. **Termination:** If ($Goals == \phi$) return $<A,O,L,\theta>$;
2. **Goal reduction:** Remove a goal $<G, A_c>$ from *Goals* such that $f(G)$ has the smallest value;
   (a) If G is $g_1 \& g_2 \& \ldots \& g_n$, post each $<g_i,A_c>$ $(i = 1, \ldots, n)$ to *Goals* and then go to 2;
   (b) If G is $(g_1 | g_2 | \ldots | g_n)$,
   For $i = 1, \ldots, n$, if $((r = \textbf{planning}(<A,O,L,\theta>, Goals \cup \{<g_i, A_c>\})) \neq null)$ return r;
   return *null;*
   (c) If G is a universally quantified goal, post $<\gamma(G),A_c>$ to *Goals*, where $\gamma(G)$ is the universal base of G, and go to 2;
   (d) If G is a literal and $\exists <A_q,Q,A_c> \in L$ such that $G\bullet\theta == (\neg Q)\bullet\theta$, return *null*;
   If there exists an access method M and $\exists \rho \in mgu$ such that $(G\bullet\theta)\bullet\rho == M\bullet\rho$, then return **planning**$(<A,O,L,\theta\bullet\rho>, Goals)$;
3. **Operator selection:** *Ops* = *candidates*(G); The ith action in *Ops* is denoted by o_i; Let $i = 0$;
4. Do {if $(++i > |Ops|)$ return *null*;
   } while ($o_i < A_c$ is inconsistent with O || ***achieve***$(o_i,G,Q,\theta,\delta) ==$ *false*);
5. Let $\theta' = \theta\bullet\{<u,v> | <u,v> \in \delta \wedge$ u, v not universally quantified variables$\}$;
   $O' = O \cup \{o_i < A_c\}$;
   For each action A_p in A,
   if (**no_threat_or_protectable**$(A_p,<o_i,G,A_c>,\theta',O',Goals) ==$ *false*) go to 4;
6. **Enable new actions and effects:** $A' = A$; $Goals' = Goals$;
   If $(o_i \in A)$ go to 7;
   $A' = A \cup \{o_i\}$; $Goals' = Goals' \cup \{<(pre(o_i)\bullet\theta)\bullet\delta, o_i>\}$; $O' = O' \cup \{A_0 < o_i < A_\infty\}$;
   add *noncode*(o_i) into θ'; if (θ' is inconsistent) go to 4;
   If *conditional*(o_i,e,Q) is *true*, post $<(condgoal(e)\bullet\theta)\bullet\delta, o_i>$ to *Goals'*;
   For each link $<A_j,P,A_k>$ in L,
   if (**no_threat_or_protectable**$(o_i,<A_j,P,A_k>,\theta',O',Goals') ==$ *false*) go to 4;
7. If *conditional*(o_i,e,Q) is *true* and e has not already been used to establish a link in L, add $<(condgoal(e)\bullet\theta)\bullet\delta, o_i>$ into *Goals'*;
8. **Recursive invocation:** If $((r = \textbf{planning}(<A',O',L \cup \{<o_i,G,A_c>\},\theta'>,Goals')) \neq null)$ return r; else go to 4;

Algorithm 2: **no_threat_or_protectable**$(A_t,<A_i,P,A_j>,\theta,O,Goals>$

1. If ($A_i < A_t < A_j$ is inconsistent with O) return *true*;
2. If ***achieve***$(A_t,\neg P,Q,\theta,\tau)$ is *false* or τ contains the bindings of existentially quantified variables, return *true*;
3. If $\{A_j < A_t\}$ is consistent with O, let $O = O \cup \{A_j < A_t\}$ and return *true*;
4. If $\{A_t < A_i\}$ is consistent with O, let $O = O \cup \{A_t < A_i\}$ and return *true*;
5. If *conditional*(A_t,e,Q) is *true*, post $<(\neg condgoal(e)\bullet\theta)\bullet\tau, A_t>$ to *Goals*, return *true*;
6. Return *false*;

5 Conclusions

FLIPPER is a goal-driven automated system management prototype. MBR is a more promising approach to fault diagnosis than other knowledge based methods. Writing pre-compiled scripts or hard-coded programs to configure/rectify faults in a managed system is inflexible. Planning is suitable for automatically generating a sequence of primitive actions to rectify diagnosed faults. In this paper, a goal-directed model-based approach to fault diagnosis and a heuristic UCPOP, are presented.
By participating in the development of FLIPPER prototype, we learn some lessons:

- The success of some AI application depends on how much users understand the underlying technology and how well AI researchers collaborate with domain experts.
- The acceptance of a particular AI technique by users needs some time and requires several demonstrations of solving a few practical problems.
- The used technology should be practical and less risks.
- Solutions should be provided incrementally from simple to complicated ones.

Acknowledgement

I would like to thank these people for their help: Alan Hydes, Claudio Bartolini, Adrian Baldwin, Patrick Goldsack, Jeremy Carroll at Hewlett-Packard Laboratories, Bristol. I wish to acknowledge the anonymous reviewers of this paper for their helpful comments.

References

[1] J. Allen, J. Hendler, and A. Tate, (Eds.) *Readings in Planning,* Morgan Kaufmann, San Mateo, CA, August 1990.

[2] A. Baldwin, C. Bartolini, G.D. Vitantonio, K. Eshghi, A Novel Algorithm for Fault Diagnosis in Internet-based Services, *Workshop on OVUA*, 1997.

[3] A. Barrett and D.S. Weld, Partial order planning: Evaluating possible efficiency gains, *Artificial Intelligence,* Vol. 67, No. 1, 71-112, 1994.

[4] D. Chapman, Planning for Conjunctive Goals, *Artificial Intelligence,* Vol. 32, No. 3, 333-378, 1987.

[5] J.S. Chen and S.N. Srihari, Candidate Ordering and Elimination in Model-Based Fault Diagnosis, In Proc. Of the 11th IJCAI, 1363-1368, 1989.

[6] K. Currie and A. Tate, O-Plan: the Open Planning Architecture, *Artificial Intelligence,* Vol. 52, No. 1, 49-86, 1991.

[7] R. Davis, Diagnostic reasoning based on structure and behaviour, *Artificial Intelligence,* Vol. 24, No. 1, 347-410, 1984.

[8] W. Hamscher, L. Console and J. de Kleer, (Eds.) *Readings in Model-Based Diagnosis*, Morgan Kaufmann, San Mateo, CA, May 1992.

[9] M.O. Hofmann, Model-Based Diagnosis Directed by Heuristic Search, *The Ninth Conference on Artificial Intelligence for Applications*, 197-203, March,1993.

[10] J. de Kleer, Focusing on Probable Diagnoses, In *Proc. 9th National Conf. On Artificial Intelligence,* 842-848, July 1991.

[11] J.de Kleer, A.K. Mackworth, R. Reiter, Characterizing Diagnosis and Systems, *Artificial Intelligence,* Vol. 56, 197-222, 1992.

[12] J.de Kleer and B.C. Williams, Diagnosing multiple faults, *Artificial Intelligence,* Vol. 32, No. 1, 97-130, April 1987.

[13] J.S. Penberthy and D.S. Weld, UCPOP: A Sound, Complete, Partial Order Planner for ADL, In *Proc. 3rd Int. Conf. On Principles of Knowledge Representation and Reasoning,* 103-114, October 1992.

[14] M. Peot and D. Smith, Threat-removal strategies for partial-order planning, In *Proc. 11th Nat. Conf. On A.I., 492-499, June 1993.*

[15] R. Reiter, A Theory of Diagnosis From First Principles, *Artificial Intelligence*, Vol. 32 No. 1, 57-96, 1987.

[16] L. Schubert and A. Gerevini, Accelerating partial order planners by improving plan and goal choices, In *Proc. Of the 7th IEEE Int. Conf. On Tools with Artificial Intelligence.* November 1995.

[17] D.S. Weld, An Introduction to Least Commitment Planning, *AI Magazine,* Vol. 15, 27-61, Winter 1994.