# Building an Expert System for Dispatch Management: Reasoning about Action Approach

M. Prokopenko[1], C. Lindley[1], V. R. Kumar[2], W. Y. Wong[1]

[1] CSIRO Division of Information Technology
Locked Bag 17, North Ryde, NSW 2113,
Australia

[2] Fujitsu Australia Software Technology
14 Rodborough Road, Frenchs Forest,
NSW 2086, Australia

**Abstract**

The task of efficient vehicles dispatching in real time becomes very important for service and transportation companies. In general, dispatch management requires complex perceptual and cognitive abilities applied under a high level of stress; the development of a knowledge-based expert system for dispatch management reduces both stress and risk in decision-making and increases economic efficiency. This paper describes a proposed object-oriented explicit consequences approach to domain knowledge representation and reasoning about domain experts' actions, and presents the prototype of a knowledge-based system developed for a manufacturing company. The key benefits of an operational logistic system for dispatch management are discussed as well.

## 1 Introduction

This paper compares different approaches to domain knowledge representation and reasoning about domain experts' actions and describes the development of a prototype knowledge-based system (KBS) which demonstrates key benefits of an operational logistic system for dispatch management.

The problem experienced by a large manufacturing company is that of the dispatch of service vehicles and personnel to service requests based upon location, service expertise, and the availability and location of spare parts. As mentioned in [13] the development of an expert consultant for an advanced computer-aided dispatching system is essential because the task of dispatching vehicles is important for the business cost-effectiveness and service productivity and at the same time "requires complex perceptual and cognitive abilities that are often applied under a high level of stress".

The developed prototype KBS supports decision-making process for a particular technical product range. There are about 50 different products in the product group and approximately 10,000 installations in the Sydney metropolitan area. These machines are serviced by number of technicians covering the city and the four regions of the Sydney metropolitan area.

## 2 The Problem Description. Key Players and Service Calls Operational Procedures

*Technicians.* Each technician is trained to service a range of products and is allocated a region where s/he will do the servicing. The technicians have to move from region to region if asked to do so.

*Dispatchers.* Dispatchers are responsible for allocating jobs to technicians. They are the ones who attend phone calls from customers and allocate jobs. It is the responsibility of each technician to communicate directly with a dispatcher to get new jobs, provide the status of current job, etc.

*Customers.* There are three type of customers, namely, contract/warranty customers, time and material customers, and other. Contract customers are given top priority in service calls.

*Request for Service*. The first step in the process occurs when a customer rings the Service to request a service call. The dispatcher enters the details for the current call, goes through the jobs in progress and waiting and gives the customer an estimated time of arrival, based on existing workloads for a technician.

*Allocating Calls to Technicians.* The dispatcher works with a board which has technician's names, grouped by region, and columns of card slots under each name. Once a call is recorded, a card with the call details is printed and placed in a slot under the name of a technician who is trained to service the particular machine. The dispatchers allocate calls to a technician within the same region. The cards are usually placed in chronological order.

*Prioritising Service Calls*. Under normal operating conditions each technician has a queue of jobs which are carried out on a First-Come-First-Served (FCFS) basis. The dispatcher may override the usual FCFS operation depending upon the priority of each call. The dispatchers change the priority of calls by changing the position of the call card in the technician's job queue or by switching the call to another technician in the region.

*Return Problem.* The technician may carry out some initial work on the machine, possibly using some parts from the van kit, and discover that extra parts are needed to fix the machine. The technician then reports this problem (RP) and requests delivery of the parts by Courier service. The technician can utilise the waiting time for the parts to arrive to carry out the next job and then return to fix the RP job. The job allocation in this case is decided by the time taken to go to the location to carry out the job, and the type of problem. If a still functioning machine has the RP (the call code is 1), then the technician leaves for the new job regardless of the waiting time. Otherwise, if the machine is down (the call code is 3), and if there is no job that can be done within the waiting time, then the technician has to wait for the part to arrive.

# 3 Knowledge Engineering

The problem is a complex optimisation or efficiency improvement problem. However, finding improved operational procedures may require techniques other than mathematical optimisation methods, eg. reasoning about action techniques or adaptive techniques that improve with experience over time. To clarify the nature of the problem and to explore the need for, and potential success of, various knowledge engineering techniques, the following activities have been conducted: 1) knowledge acquisition; 2) knowledge representation and engineering; 3) development and verification of the prototype KBS.

The knowledge required for building the KBS was acquired through a series of interview sessions. During the meetings, the range of the problem and service calls operational procedures were identified. The prototype KBS has been designed to take into account the questions that need to be resolved before an allocation can be done/recommended, and the decisions made within the system include but are not limited to the following:

- when a request is received, which service van should it be given to;
- which technicians are currently in the region of the request;
- which technicians are busy;
- how long will busy technicians require to complete their current jobs;
- which technician will be available earlier;
- which technicians are closest to the source of the request;
- which technicians have the appropriate expertise;
- which technicians have access to the necessary spare parts either by carrying them or being close to a suitable store;
- what is the estimated response time;
- what is the expected time to deliver the necessary parts;
- what jobs can be done during the delivery (RP) time by a particular technician;
- should a technician with RP wait at the current site or move to the next available job;
- what job has to be allocated to a technician who completed current job.

## 3.1 Reasoning About Action Approach.

The domain can be characterised as *a dynamical system* or as "a system whose state changes over time, and where effects flow forward in time so that the non-input part of the state at one time can only depend on its earlier states" [17]. The agents or key players of the system are capable of performing *actions* that influence state variables and thus change the system state. The task of next state prediction has become a major focus of research in Reasoning about Action and in related areas of Artificial Intelligence such as Belief Change. It can be informally described as follows: "given information about the current state of a dynamic world as well as information about the occurring event, we want to derive information about the next state of the world" [12].

The historical approach to formalising reasoning about change has been to represent domain knowledge declaratively in a formal language capable of inferring "that a certain strategy will achieve its assigned goal" [11]. Traditionally logic has been chosen as a representation language and various reasoning systems[1] have been designed to address the problem. The monotonic situation calculus [11] and developed nonmonotonic logics such as default logic [15], or circumscription [9, 10, 6] try to *infer* what facts are true once the events have occurred (actions have been performed) and thus answer queries about the theory without actually updating it. However, "to determine what is true about the world after an action has been performed, the default frame axiom must be examined once for every fact of interest" [3] in such an approach leading to serious computational problems. An alternative way to formalise reasoning about change was

[1] The term "reasoning system" is used to refer to "any formal system that produces inferences about the effects of events" (Peppas, 1993).

proposed in the STRIPS approach [1] and extended in the Possible Worlds Approach (PWA) [3], and the Possible Models Approach (PMA) [18]. The common to the approaches idea is to update a single model (theory) of the world transforming it into the "minimally-changed" one in which the consequences of the actions under consideration hold. Consider the STRIPS approach representing world models by sets of first-order logical formulas. A STRIPS system describes an action by its precondition (the applicability condition expressed by a first-order formula), add list and delete list (the lists of formulas that must be added to or deleted from the current world model respectively) [1, 7]. This approach to action representation has two clear advantages: an effective solution to the frame problem [11] - there is no need at all to list what persist through time since
procedure updates only directly affected facts, and impossibility to generate multiple extensions of the Yale Shooting Problem type [5] - actions are described by incremental changes to a database that keeps a single model (theory) of the world. The major problem on this way is, however, that the STRIPS approach fails to solve the ramification problem [2]: "all consequences of an action must be explicitly listed in the add and delete lists" [3].

In order to estimate the limitations of the STRIPS approach with respect to the domain and the domain's complexity we described the domain actions: allocate(*tech, new_job*), finished_job(*tech, job*), rp_report(*tech, job*), rp_allocation(*tech, job*), etc. as triples (P, D, A), where P is a precondition, and D and A are delete and add lists respectively - following [7]. For example, the following operator (P, D, A) describes the case of the action rp_allocation(*tech, job)* when a job with the RP for a malfunctioning machine (code 3) is re-allocated to a technician with at least one waiting job and the courier delivery time is sufficient to conduct some waiting job.

Precondition P[2]:

current_job(*tech, job*) & job_status(*job*, "rp_3") &
status(*tech*, "busy") & waiting_jobs(*tech, list*) &
¬**empty**(*list*)[3] & location(*tech, current_loc*)&
**clock**(*time*) & delivery(*job, delivery_time*) &
∃ *next_job*(
**member**(*next_job, list*)[4] & job_location(*next_job, new_loc*) &
repair(*next_job, repair_time*) & **travel**(*current_loc, new_loc, travel*) &
(*repair_time + 2 * travel < delivery_time*) & **delete**(*next_job, list, new_list*)[5]
)

Delete list D:

current_job(*tech, job*)
waiting_jobs(*tech, list*)
location(*tech, current_loc*)
job_start(*tech*, _ )
job_finish(*tech*, _ )

Add list A:

current_job(*tech, next_job*)
waiting_jobs(*tech, [ job | new_list]* )
location(*tech, new_loc*)
job_start(*tech, time + travel*)
job_finish(*tech, time + travel + repair_time*)

The first conclusion we can now make is that indeed the STRIPS leads to exponential difficulties as discussed in [3] due to the need "to split the original action into a variety of special cases, specifying preconditions, add lists and delete lists for each". However, the following observations should be made:

- the initial state of the domain is complete and determined by the appropriately initialised facts on technicians, and by the given travel times between suburbs;
- the system consisting of the initial state and the operator descriptions satisfies the Soundness Theorem [7];
- all actions are deterministic and do not introduce incomplete information into the theory;
- the update procedure is *symmetric*: every deleted fact has an added counterpart for every considered case.

The PWA, instead of describing explicit consequences of all actions, inferentially investigates domain constraints exploiting the idea that many

[2] Assuming a standard unification of variables (denoted by strings in italic), where _ is a standard anonymous variable (M.4 User's Guide, 1991).
[3] Strings in bold denote function calls rather than fact/relation names.
[4] Assuming straightforward definition of the **member**(*x,y*) function: returns True if *x* is a member of *y*.
[5] Assuming that the function **delete**(*x,y,z*) deletes element *x* from the list *y*; *z* is the resulting list.

apparently distinct preconditions and delete lists "may in actuality be manifestations of a small number of domain constraints" [4]. We do not intend here to formally apply systematic methodology for the assessment of various logics of actions [17] but the domain is likely within the range of applicability of the PWA (and, therefore, the broader PMA). In other words, it is not impossible to find a set of domain constraints always producing desirable inferences in the domain. However, complexity of this task is comparable with that of maintaining satisfactory descriptions of the actions in the STRIPS [14]. Moreover, a knowledge-based system incorporating the PWA requires implementation of an appropriate theorem prover and a tool for proofs' analysis. Computational issues related to the design of such generic formal tools are not completely resolved at the moment.

### 3.2 The Object-oriented Explicit Consequences Approach.

Although in general the PWA leads to a more compact knowledge representation it does not reduce the time needed to evaluate the result of any particular action if compared with the STRIPS approach [3]: the STRIPS operator descriptions in some sense "compile" the interactions between the actions and the domain constraints into the delete list. The mentioned symmetric character of the considered update procedure for the domain is the reflection of such an interaction. This feature can lead to the obvious simplification: it is no longer necessary to keep delete lists in the explicit form since the update procedure can add appropriate facts *instead of* their predecessors if the precondition holds. To justify the assumption that every deleted fact has an added counterpart for every considered case we observe an *object-oriented* character of the facts in operator descriptions: the update procedure is *symmetric* because the *slots'* values of the *instances* of the corresponding *classes* are to be changed.

The object-oriented character of the domain knowledge not only explains why it was possible to obtain symmetric delete and add lists but allows also clarify the ontological characteristics of the domain actions. In particular, there is a distinction between structural ramification when "the action can affect features of other objects than those which occur as arguments of the action" and local ramifications involving only "features of the argument objects" [17]. The described actions avoided any structural ramifications or, in other words, affected only features (slots) of objects (instances) that were action's arguments. Nevertheless, other scenarios are plausible as well. For example, consider the action rp_allocation(*tech, job*). In some cases the job, *job* is appended to the beginning of the waiting list of the technician *tech*. This might as well shift the last job in the list beyond the limit of the response time. That in turn can lead to re-allocation of this job to another technician affecting *another object* of the system. Thus, the distinction between structural and local ramifications in the domain becomes clearer under object-oriented representation.

Finally, we can represent every operator description in the form:

$$
\begin{aligned}
&\exists\ \textit{Technician} \in \text{technician(id, job, queue, location, status, job\_start, job\_finish)} \quad \&\\
&\exists\ \textit{Job} \in \text{call\_job(number, location, repair\_time, delivery\_time, code, courier\_arrival)}\\
&\ni\\
&\textit{Technician}(\textit{tech, job\_x, waiting\_jobs\_x, location\_x, status\_x, start\_x, finish\_x}) \quad \&\\
&\textit{Job}(\textit{job, location, repair\_time\_y, delivery\_time\_y, code\_y, courier\_arrival\_y}) \quad \&\\
&\text{Precondition}(\textit{Technician, Job}) \equiv \text{True}\\
&\Rightarrow \qquad (1)\\
&\textit{Technician}.\text{job} = \textit{job\_X}, \quad \textit{Technician}.\text{queue} = \textit{waiting\_jobs\_X}, \quad \textit{Technician}.\text{location} = \textit{location\_X},\\
&\textit{Technician}.\text{status} = \textit{status\_X}, \quad \textit{Technician}.\text{job\_start} = \textit{start\_X}, \quad \textit{Technician}.\text{job\_finish} = \textit{finish\_X},\\
&\textit{Job}.\text{repair\_time} = \textit{repair\_time\_Y}, \quad \textit{Job}.\text{delivery\_time} = \textit{delivery\_time\_Y},\\
&\textit{Job}.\text{code} = \textit{code\_Y}, \quad \textit{Job}.\text{courier\_arrival} = \textit{courier\_arrival\_Y},
\end{aligned}
$$

where $\Rightarrow$ means that if the antecedent is satisfied (ie. there exist instances of the "technician" and "call_job" classes such that their slots evaluate under standard unification the Precondition(*Technician, Job*) as true) then all variables ending with _X and _Y differing from their counterparts ending with _x and _y substitute them in such instances. For example, the considered case of the rp_allocation(*tech, job*) action can be represented as:

$$
\begin{aligned}
&\exists\ \textit{Technician} \in \text{technician(id, job, queue, location, status, job\_start, job\_finish)} \quad \&\\
&\exists\ \textit{Job} \in \text{call\_job(number, location, repair\_time, delivery\_time, code, courier\_arrival)}\\
&\ni
\end{aligned}
$$

*Technician*(*tech, job, [ head | tail], location,* "busy"*, start, finish*) &
*Job*(*head, new_location, repair_time, delivery_time,* "rp_3"*, courier_arrival*) &
**clock**(*time*) & **travel**(*location, new_loc, travel*) & **(***repair_time + 2 * travel < delivery_time***)**
⇒
*Technician*.job = *head,* *Technician*.queue = *[job | tail],*
*Technician*.location = *new_location,* *Technician*.job_start = *time + travel,*
*Technician*.job_finish = *time + travel + repair_time,*

where the call_job instance remained unaffected.

The actions descriptions in the form (1) that eliminates the delete lists, allows only local ramifications and guarantees soundness of operators describing actions justify the conclusion that the STRIPS approach enhanced with object-oriented knowledge representation is appropriate for the domain.

## 4 Implementation of the Prototype KBS

The M.4 expert system shell (version 2.0)[6] and Toolbook (version 1.53)[7] have been used to design the prototype KBS. Currently, the prototype KBS is implemented under Microsoft Windows (version 3.1)[8] platform on an IBM PC 486 and includes the knowledge-based system; a database module with information on technicians' attributes; and graphical user interface.

The KBS was developed via rule-based and object-oriented programming approaches. The domain has been classified into two different classes representing entities of "technician" and "request for job". The KBS is composed of objects instantiated from these classes, IF-THEN pattern matching rules, and procedures defined on the instances. The prototype KBS supports dynamically created instances of the "Technician" class and controls instances of requests ("Call_job"). An example of a rule in the KBS is given below:

```
if
      call_job <- init = CALL          and
      search_local_for(CALL) = X       and
      handle(X,CALL) = TECH            and
      allocate(TECH,CALL)
then
      action(process_customer_call) = continue.
```

The KBS supports virtual time operations with granularity of one minute and allows to select one of the following modes: "process customer call", "answer technician call", "display all waiting calls", "report technician status", "finish", covering different scenarios of dispatchers' work. During runtime, the KBS retrieves static information from the database, handles cyclical control of incoming customers' and technicians' calls, and allocates new and waiting jobs to available or suitable technicians. The KBS demonstrates realistic situational awareness in terms of events and actions.

[6] M.4 is a product of Cimflex Tecknowledge Corp.
[7] Toolbook is a registered trademark of Asymetrix Corp.
[8] Microsoft Windows is a trademark of Microsoft Corp.

## 5 Summary

Verification of the prototype of KBS involved program debugging, error analysis, input acceptance and output generation, reasonableness of operations and run time control checking, and allowed to distinguish the following features and benefits. The KBS is capable of handling different situations in dispatch management within a real (virtual) time environment; provides realistic decision support model and consistent recommendations; prototypes an operational system for improved service and dispatch control which may replace paper system to support staff; recalculates best known solution every time there is a change (a new service request, a completion or RP report, etc.) and produces a fast response; and constitutes a platform for rigorous testing of alternative models (strategies) in simulation for decision improvement.

### Acknowledgements

The authors are thankful to Pavlos Peppas for his suggestions and comments on different paradigms of Reasoning about Action, and to Charles Chung for his contributions towards building a graphical user interface for the prototype KBS.

## References

1. R. Fikes & N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence*, 2, pp. 189-208, 1971.
2. J. J. Finger, *Exploiting Constraints in Design Synthesis*. PhD Thesis, Stanford University, Stanford, CA, 1987.
3. M. L. Ginsberg & D. E. Smith, "Reasoning about Action I: A Possible Worlds Approach", *Artificial Intelligence*, 35, pp. 165 - 195, 1988.

4. M. L. Ginsberg, & D. E. Smith, “Reasoning about Action II: The Qualification Problem”, *Artificial Intelligence*, 35, pp. 311 - 342, 1988.
5. S. Hanks & D. McDermott, “Nonmonotonic Logics and Temporal Projection”, *Artificial Intelligence*, 33, pp. 379 - 412, 1987.
6. V. Lifschitz, “Computing Circumscription”. In *Proceedings International Joint Conference on Artificial Intelligence*, pp. 121 - 127. Los Angeles, 1985.
7. V. Lifschitz, “On the Semantics of STRIPS”. In *Proceedings of the 1986 Workshop on Planning and Reasoning about Action*, Timberline, OR, pp. 1 - 9, 1986.
8. M.4 User's Guide, *A Guide to Developing and Delivering Knowledge Systems*, Cimflex Tecknowledge Corporation, 1991.
9. J. McCarthy, “Circumscription - a Form of Non-monotonic Reasoning”, *Artificial Intelligence*, 13, pp. 27 - 39, 1980.
10. J. McCarthy, “Applications of Circumscription to Formalizing Common-sense Knowledge”, *Artificial Intelligence*, 28, pp. 89 - 116, 1986.
11. J. McCarthy & P. Hayes, “Some Philosophical Problems from the Standpoint of Artificial Intelligence”. In *Machine Intelligence*, vol. IV, edited by B. Meltzer & D. Michie, pp. 463 - 502, 1969.
12. P. Peppas, *Belief Change and Reasoning about Action. An Axiomatic Approach to modelling Inert Dynamic Worlds and the Connection to the Logic of Theory Change*. PhD thesis, Dept. of Computer Science, University of Sydney, 1993
13. J-Y. Potvin, Y. Shen, G. Dufour, J-M. Rousseau, “Learning Techniques for an Expert Vehicle Dispatching System”, *Expert Systems with Applications*, Vol. 8, No. 1, pp. 101-109, 1995.
14. M. Prokopenko, C. Lindley, V. R. Kumar, “The Application of Reasoning about Action Techniques to Dispatch Management”. In *Proceedings of the AI’95 First Australian Workshop on Commonsense Reasoning*, Canberra, pp. 74 - 88, 1995.
15. R. Reiter, “A Logic for Default Theory”, *Artificial Intelligence*, 13, pp. 81 - 132, 1980.
16. R. Reiter, “Nonmonotonic Reasoning”, *Ann. Review of Computer Science*, 2, pp. 147 - 186, 1987.
17. E. Sandewall, *Features and Fluents. A Systematic Approach to the Representation of Knowledge about Dynamical Systems*. Research Report. Dept. of Computer and Information Science, Linkoping University, Sweden, 1994.
18. M. Winslett, “Reasoning about Action Using a Possible Models Approach”. In *Proceedings of AAAI-88*. St. Pauls, MS, pp. 89 - 93, 1988