Proof Planning and Configuration

Proof Planning & Configuration

Helen Lowe

Department of Computer Studies, Napier University, Craiglockhart, Edinburgh EH14 1DJ

Michal Pechoucek, & Alan Bundy

Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh

1. Introduction

This paper presents two configuration problems: that of configuring computer hardware to meet a given specification, and an "engineer-and-made-to-order" problem in the domain of breathing air compressor production. We demonstrate how the different kinds of knowledge needed to solve problems in each domain (which we label *factual*, *heuristic*, and *strategic*) was captured for a *proof planning* system. The systems perform favourably when compared with human experts. The ideas developed for the first domain (computer hardware) transferred remarkably well to the new domain (compressors), the time taken from knowledge acquisition to prototype being less than three months. We show that such systems are also easy to maintain, and to adapt to similar but distinct problems in the same domain. We end with suggestions of further domains and problems which we believe are amenable to the proof planning approach.

2. Two configuration problems

2.1 Computer Hardware

Configuration is a task involving much attention to detail and some search problems. For example, by getting it wrong a company can lose a great deal of money and goodwill. It is easy to make mistakes, *e.g.* by forgetting vital cables or not realizing the most efficient configuration.

A common business practice is for a company (we shall refer to this as the client company) to issue a specification and to invite tenders for a configuration. The tendering companies each produce descriptions of candidate configurations meeting this specification, together with details of capabilities and costs. The client company will not necessarily accept the lowest tender (in terms of absolute cost) but may aim to accept the tender which proffers the best "value for money". (This is a consequence of the fact that the specification may be no more than a minimal requirement.)

Once the tender is accepted, a contract is signed between the client and the successful tenderer. This contract will state that the computer firm agrees to provide a system based on the proposed configuration which at least meets the given specification for the agreed cost. This means that, if the configuration turns out to be deficient, the computer firm must make good the shortfall and bear any extra cost involved. In many ways, the loss of goodwill, from the attendant delays in providing the promised service is, in the long term, even more expensive for the firm involved.

Since it is so important, one might wonder why mistakes occur with such frequency. The answer seems to be a combination of rapidly changing product lines, and much low-level detail which is difficult to absorb. Often the computer firm's representative is an expert in sales at the higher level of specification, but cannot be expected to keep track of the low-level configuration details.

In more precise terms, the problem is in the generation of unsound "solutions", *i.e.* the objects which are generated are not *legal* objects (configurations). They do not obey the laws of configuration, *e.g.* that every device in the configuration should have a legal connection.

For the hardware configuration problem, an interactive system was not required, in that the specification was given as completely as the user wished at the start of a session. The system was required to present solutions at a high level of detail (in terms of devices and function rather than cables and connections) until the user was satisfied; the detailed configuration for the user's choice would then be given.

Such a system was produced as a prototype, using Quintus Prolog. It proved successful not only for achieving the given task, but also as a proof of concept for proof planning as a method (Lowe, 1994)

2.2 Compressors

CompAir Reavell Ltd, a member of the Siebe group, manufacture high pressure gas compressors mainly for the naval, NGV and breathing air markets. There are three stages of quotation engineering. A rough customer specification goes to the sales department, where preliminary sizing is carried out. The main task of the sales people is to ascertain customer needs, ask various questions, to determine which construction group the produce belongs to, and to offer approximate budgetary guidelines. The next stage is the quotation stage, at the end of which a complete product number is produced which fully specifies the solution from a construction point of view: drive, fitting, filtering, *etc*. The quotation people reason on the basis of past experience and intuition, browsing a huge database of components. In the third stage, the product number is passed on to the construction department, who decide whether the product can be constructed. If it is (and it may not be), they produce a drawing of the device to be manufactured which is sent to the workshop.

Our objective was to produce an automatic configurer for specifying compressors, which would then be a designed and constructed afterwards - the *engineer-and-made-to-order* type of configuration described above. The system was required to present a logically sequenced order of questions, together with all legal options, to the user. The system should then price the solution, create the construction description number, and set up the final quotation document.

We see that with this domain, the client wanted an *interactive* system, with the user presented with legal choices at each of the main stages. This was the main new interest in extending the proof planning concept in capturing the knowledge for this domain.

Also, given the problem of rapidly-changing product lines, a prime goal was for the system to be readily maintainable. It was believed that the particular formalism chosen, expressed in logic and implemented in Prolog, with its separation of knowledge and control, would facilitate the maintenance of *all* types of knowledge in the system. To this end, a prototype system was build and underwent field testing at Compair.

3. Types of knowledge

3.1 Problem solving

Humans use their knowledge to solve problems. In many tasks we observe that the "knowledge" is of different kinds. We speak of "knowing that", or "knowing how". We use factual knowledge, which is immutable, and we use "rules of thumb", which may grow with our experience, or change. We have guidelines, and also the knowledge that tells us when it is safe to break those guidelines.

Many knowledge based systems suffer from the difficulty of separating these different kinds of knowledge. Knowledge bases are compiled with the aid of human experts who are often unable to unravel them, or even understand the distinction. As new objects are added or new rules discovered it becomes increasingly difficult to maintain the system. Trying to hone systems so that they are efficient , by the use of control information, or to incorporate design considerations in more open-ended problems, only exacerbates the difficulties in trying to ensure the resulting system is reliable, robust, and maintainable.

Our systems are based on such a separation, enabling each type of knowledge to be encoded declaratively. This is of the utmost importance, if we are to be able to check that the logical formalism given accords with our understanding of the semantics. We need to be able to check, separately and independently:

- That the facts represented are "true", or at least what we intend
- That procedures are captured correctly.

We have identified three main kinds of knowledge. We explain these, with examples, in the next three sections.

3.2 Factual Knowledge

There are many facts to be recorded in each domain. For example, in the hardware configuration domain there are catalogues, price lists, and manuals. We record matters of fact about devices such as

> "XXX is a disk drive."
>
> "The component XXX has a capacity of 571 Mb."

Others correspond to the "axioms" of a domain, for example

> "Every device in a configuration must have a means of connection."

It is clear that object level knowledge must be updated as new products come into being and others become obsolete, thus ease of maintenance is an issue.

3.3 Strategic Knowledge

If factual knowledge is "knowing that" then strategic knowledge is "knowing how". Faced with a problem, there may be several steps that we need to undertake in order to solve it. It is unlikely that a random choice of tasks in any order will be successful, or at the very least it will not be efficient. By talking to experts in each domain, we were able to elicit the strategic knowledge they use. For example:

> " When configuring a hardware system from scratch it is best to follow steps in this order: choose processor, then configure disk drives, then assess backup needs."

Such knowledge changes less often than factual or heuristic knowledge, for example if whole structures or new kinds of products are added or altered. Explicitly and separately held control knowledge enables us to update it on these occasions. In addition, the separation of control knowledge also means that the system can be generalized and used for other, similar tasks using the same basic knowledge base.

3.4 Heuristic knowledge

Heuristic knowledge often arises from the *soft constraints* of a system: it may be desirable but not absolutely necessary to keep within such constraints. Heuristic knowledge is, or may be, changing with time and circumstance; for instance, the fact that particular configurations lead to inefficiency may only be learned from experience of actual running configurations. Examples of such knowledge are

> "It is best to have fewer than the legal maximum of four disk drives on a channel."
>
> "Do not connect the system disk drive and the X model of printer to the same channel".

It may have been discovered - or even anticipated - that violation of these rules leads to a system running inefficiently, for example, rather than it would not run at all.

4. Approaches to modelling

4.1 Traditional

Production rule systems, although popular, are traditionally bedevilled with maintenance problems, XCON (see, for example McDermott, 1982) being a prime example. A full account may be found in Lowe (1993). The addition of a new product line may necessitate many changes, and AI expertise is needed to make these changes. Our domains both have rapidly changing products and ease of maintenance is a priority. We therefore rejected this approach.

For the compressor domain, we carried out knowledge analysis within the KADS methodology in order to compare this approach with that of proof planning. In our judgment, KADS is designed for larger scale projects than that of this domain. Knowledge orientation within proof planning is considerably more natural for the people maintaining and enhancing the knowledge base than the complex model layering in KADS. Details of this may be found in Pechoucek (1996).

4.2 Proof planning

Bundy (1988) proposed that a technique known as *proof planning* , which was originally developed for use in theorem proving and program synthesis, could be used for IKBS general. This technique makes use of methods written in a meta-logic to facilitate the task of writing proof plans to guide the search for proofs in automatic theorem proving, and possesses the following properties:

- Efficiency, because the combinatorial explosion is avoided, or at least greatly mitigated.
- Generality, because a proof plan may be applicable to many proofs.
- Maintainability, because the separation of factual knowledge from control knowledge means that either may be changed without affecting the other.
- Explanatory power, because control decisions can be explained at the appropriate level, rather than by generating long chains of low-level choice points in the inference process.

These properties are important for *any* knowledge based system. Thus proof plans can provide a useful vehicle for expressing strategies for problem-solving in other domains, including non-mathematical ones.

The global strategies for carrying out configuration tasks are expressed as *proof plans*. In the context of configuration, we can think of proof plans as the expressions of meta-level strategies. We can go further. High-level strategies for configuration can be developed, and encapsulated as proof plans, or supermethods.

The proof planning system fits together *tactics* in a flexible. A tactic is a program encapsulating a significant proof step with its attendant lower level steps. The latter are typically ones of less interest to the user of the system, and correspondingly harder to keep track of. We would prefer these to be taken care of automatically so that we can concentrate on the “interesting” steps.

A method is a specification for a tactic. Methods have slots for

- method name

- input , which the input goal must match
- preconditions , which are conditions which must be true of the input if the method is to be applicable
- output , which will match the rewritten input goal if the method is applied
- effects , which are conditions on this output goal if the method is applied
- tactic, the program to be applied to the goal at this point.

The nice feature of specifying tactics by methods in this way is that it models the "user" or "customer view" of configuration, as opposed to the "engineer view", which is "modelled" by actually executing the tactic. This makes developing good explanation facilities a realistic possibility. This is not true of rule or constraint based systems which work at a low level, where the search space is more complex and reasons for the choices made may not be readily apparent.

5. Development life cycle

5.1 Computer Hardware

As a proof of concept, we looked at the problem of how to synthesize a configuration which meets a specification for a computer system. The resulting system (Lowe, 1993) was able to synthesize a term representing all the components needed, together with details of the connections between them. Now we have turned our attention to another configuration problem: that of configuring compressors.

The careful separation of *object-level knowledge* from *meta-level control* and *heuristic knowledge* is an important feature of our approach. This has the benefit of facilitating maintenance, in a domain which traditionally has been bedevilled by maintenance overheads. Although other systems, notably XCON , have moved in this direction, in our configuration system this separation is both strict and explicit. Such an approach makes the task of maintaining knowledge bases more tractable and reliable.

It should be stressed that this was no easy task: knowledge bases have to be compiled with the aid of human experts who are often unable to unravel the different kinds of knowledge, or even realize the distinction. However, if this is not done, then, as new objects are added or new rules discovered, it becomes increasingly difficult to maintain the system. Thus the initial effort in unravelling these different kinds of knowledge, which is forced upon us using our approach, should in practice prove worthwhile in the long run.
We developed an axiomization of configuration theory which allows the use of theorem proving techniques. This renders a more precise definition of the problem and enables us to reason about it in a rigorous way so as to ensure soundness.

Because the search problem is, in general, quite considerable, some kind of control is needed for all but small hardware systems synthesis. We chose to encapsulate control strategies explicitly, using a meta-level language to be used in the preconditions slots of methods. The main benefit of separating object-level knowledge from meta-level knowledge in this way is ease of maintaining both kinds of knowledge.

Our main aim was to produce a prototype system which was capable of being maintained by people who did not have knowledge of the whole system. So, for example, some files would hold product information and as such would be updated as new products came on to the market and old ones became obsolete. Others would hold heuristic information and need updating by engineers from time to time as new information became available. Yet another file would hold information designed to encapsulate strategies for configuration: at the present time this holds the knowledge of configuration strategy that we have discovered and implemented so far. It can be constantly updated in the light of experience.

The knowledge is separated in such a way as to render it possible for different people to update files independently of other knowledge held by the system. Thus the addition of one new component does not render the whole configuration system obsolete. There are no hidden ``knock-on'' effects, such as those reported in systems where knowledge, control, and context are inextricably linked.

5.2 Compressors

We developed ICON (*Industrial CONfigurator*) in LPA Prolog 3.1 for Windows, running on 386 or 486 processors. Its behaviour is given by an ordered set of methods and a formalized domain theory. Ladder logic was used to formalize the object-level, or factual, knowledge about components and attributes. An ordered set of methods was used for expressing the decision processes carried out by experts and tactics were used for storing the information about how to create the particular produce number, a feature of this domain. Proof planning introduces three sequential stages in the system behaviour: planning, validation, and execution.

In the planning stage, the user is asked to give as much information as they can to give a direction to the search for possible solutions. In the validation stage, the system offers the best found solution and user either accepts this or returns the system to the planning stage to redo some decisions. In the execution phase, the system creates the product number and the final quotation document.

To illustrate the flexibility of proof planning, it was decided to implement another piece of software to address a different task within the same domain. This was the Advanced Planner. Here, users are presented with a set of attributes; they select (in turn) only those they care about and are presented with a list of legal options for each. The planner then searches through the space of solutions to find one meeting the user's criteria. Each solution found is presented to the user for inspection and validation. It is notable that the prototype system for the second task, using the domain knowledge collected for the first, was implemented in just a single day.

6. Evaluation

The hardware configuration domain can be seen as our initial proof of concept. With many rival systems around it was not expected that field testing would be done in earnest. However, the ICON system *was* designed to be used. We evaluated

- How easy it was to capture the knowledge needed in the right form, and how long this took.
- The time taken to design and implement the prototype system
- The performance of the system, tested by its potential users.
- How easy it would be for its users to maintain.

Michal spent two weeks at Compair Reavell carrying out the knowledge elicitation phase. Afterwards, it took around eight weeks to completely build and design the system. He then spent two further weeks at CompAir, introducing the tool to sales people, quotation and construction departments, and management.

ICON met all requirements and reduces the work of two days to a single minute. One addition was an explanation facility, which was readily provided, to enhance the sales-customer relationship. It was thought that, with this facility, the service could be provided direct to customers via the Internet.

A few inconsistencies were found during testing. These turned out to be easy to right, for example a redundant attribute, some misleading vocabulary, incorporation of measurement units, and a couple of rule changes.

For maintenance, Michal organized a short Prolog tutorial and explained the maintenance of the knowledge base. After the tutorial session, the IT staff were asked to refine the global knowledge base in two ways:

1. Add a new set of compressors (water-cooled)
2. Add a new attribute (weight) to the set of properties.

This took Michal (with his expert knowledge of Prolog and his own system) twenty minutes. The CompAir staff averaged thirty minutes - this was considered a pleasing result, considering the complexity of the tasks involved and the inexperience of the staff.

7. Further work

We aim to extend the work in two ways.

1. Explore heuristic knowledge in the compressor domain
2. Find new domains with new challenges for proof planning

Over the time period, no strictly heuristic knowledge was discovered - given that we count strategic knowledge as control knowledge rather than heuristic. As found with the hardware domain, such knowledge often appears over a long period. It would be interesting to track the use of the ICON system over the next year or so to see whether a need for heuristic knowledge arises.

There are many domains with tasks which can be seen as configuration - design of module courses in a university, the design of a safe and aesthetically pleasing room (kitchen, bedroom, *etc.*), and ultimately of buildings. Such software empowers the client by giving them both the expertise and the choice to design their own artifacts.

Some of these domains will prove interesting from the user interface point of view, not just in terms allowing the user to interact directly with a graphic interface (for example, to design a kitchen), but also in terms of deciding what knowledge to give the user - timeliness and appropriateness are key issues. Proof planning systems are ideal for proving good explanations, since these can be extracted via the truth or falsehood of one or more preconditions.

8. Conclusions

We conclude that the systems developed demonstrate

- Ability: to perform the tasks required efficiently, quickly, and error free
- Maintainability
- Adaptability to different tasks
- Tractability, *i.e.* quick and clean implementations of both domains.

We have developed formalizations of two configuration domain in order to apply established theorem-proving techniques to the problem of configuring hardware to meet specifications. This ensures the underlying soundness of the system, in that all solutions generated will at least be legal. Using this as a basis, we have been able to encapsulate the higher-level reasoning, used informally by human experts, in a more formal way in order to facilitate the generation of well-designed and "natural" solutions, and to generate these solutions as efficiently as possible by guiding search by means of meta-level reasoning techniques. We would not argue that we have discovered, much less implemented, all strategic knowledge brought to bear on this problem by human experts. However, the methodology of this approach enables this to be done incrementally. This is due to the fact that the meta-level knowledge is captured independently of the object-level theory. So not only can we maintain the system by the addition of new components, and even new types of components, but we can also experiment with new strategies. Given that human expert strategies are often opaque at first, there is the added benefit that we can gain new insights into the process of human reasoning in a complex task. Using the proof planning methodology, we can carry out these kinds of modifications

and experiments in a principled way which is easy to track.

Acknowledgments

We would like to thank all those at Hewlett Packard Research Laboratories and Siebe who made these projects possible. In particular, we wish to thank Angus Appleby from Compair Reavell, Robin Wilkinson from CompAir BrooWade, and Richard Bradford from Siebe, plc. We are grateful to Jim Doheny from the Artificial Intelligence Applications Institute in Edinburgh for useful discussion about KADS issues. Michal Pechoucek's studies were facilitated by the *Excalibur* Scholarship Scheme.

References

Bundy, A (1988). The Use of Explicit Plans to Guide Inductive Proofs. In Lusk, R and Overbeek, R (eds.), *9th Conference on Automated Deduction*, pages 111-120. Springer-Verlag.

Lowe, Helen (1993). The Application of Proof Plans to Computer Configuration Problems. Unpublished PhD Thesis, Department of Artificial Intelligence, University of Edinburgh.

Lowe, Helen (1994). Proof Planning: A Methodology for Developing AI Systems Incorporating Design Issues. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, August 1994.

McDermott, J (1982). XSEL: A Computer Sales Person's Assistant. *Machine Intelligence*, 10(1), 1982.

Pechoucek, Michal (1996). Industrial Configuration and Proof Planning. Unpublished MSc Thesis, Department of Artificial Intelligence, University of Edinburgh.