DECIDING PARAMETER VALUES WITH CASE-BASED REASONING

C. J. Price and I. S. Pegler

Centre for Intelligent Systems
Department of Computer Science,
University of Wales, Aberystwyth,
Dyfed, SY23 3DB, United Kingdom.
Email: cjp@aber.ac.uk

Abstract

This paper describes an industrial application of case-based reasoning (CBR) in the aluminium die-casting industry: the setting of parameters of a pressure die-casting machine for a specific die. As well as describing the application, the paper shows how the system has evolved. The use of cases has given the system a flexibility which has enabled it to be used in ways that were not foreseen at the outset of the project.

The pressure die-casting application is a good example of a common problem: the correct setting of system parameters depending on a set of input values. The paper discusses the general use of CBR for deciding on parameter values, and describes the lessons that have been learned through the construction of the die casting parameter setting system.

This paper appeared in the 1st UK Workshop on Case-based Reasoning, January 1995, and an updated version is in Progress in Case-based Reasoning, Ian Watson (ed), pp121-133, Springer Verlag 1995.

1. Introduction

This paper describes Wayland, a computer program which applies case-based reasoning [Kolodner] to the problem of setting parameter values on an aluminium pressure die-casting machine.

There have been previous applications of CBR for deciding how companies should operate machinery, notably Clavier [Hennessy and Hinkle, Hinkle and Toomey]. Wayland differs from Clavier in two important ways. Firstly, it performs approximate numerical matching to past cases, where Clavier only does exact textual matching. Secondly, adaptation of the result is an important part of Wayland, whereas it seems to have been dropped from Clavier.

Wayland has been deployed for two years, and demonstrates the clear benefits that such a system can provide. The paper describes the mechanics of the Wayland program and the foundry’s experience in using it. Finally, the paper considers the wider possibilities of applying case-based reasoning to deciding on parameter values, and draws out the lessons learned from implementing the pressure die-casting system and from observing how different types of user access it.

2. The pressure die design problem

Pressure die casting involves injecting molten metal at very high pressure into a mould (a die), where it cools to make a casting. Figure 1 illustrates many of the main objects involved in the process. Some of the key concepts are:

> **Gate.** The hole through which the molten metal enters the impression part of a die (the shape of the **casting** to be made). The gate is usually kept to a narrow slit to reduce the cost of the casting. It has to be removed from the casting, and if the gate is more than about 3mm deep, the excess has to be sawn off rather than clipped. Consequently, both the gate depth and the gate cross sectional area are of interest.
>
> **Sleeve.** The tube into which the molten aluminium is poured so that it can be pushed by the **plunger** into the die.
>
> **Tip.** The tip is the end of the plunger by which the metal is pushed into the die. Smaller tips allow higher pressures to be exerted on the die. Larger tips allow quicker filling of the die.
>
> **Number of impressions.** Some dies make more than one component per casting. For example a die which makes four components from a single casting is referred to as a four impression die.
>
> **Cycle time.** This is the total time to make a casting, from one injection of metal to the next. It includes filling the casting (cavity fill time), cooling time, and extraction of the component from the die.

The terms given here are those used in die casting of aluminium. Some terms and concepts will vary in the casting of other metals.

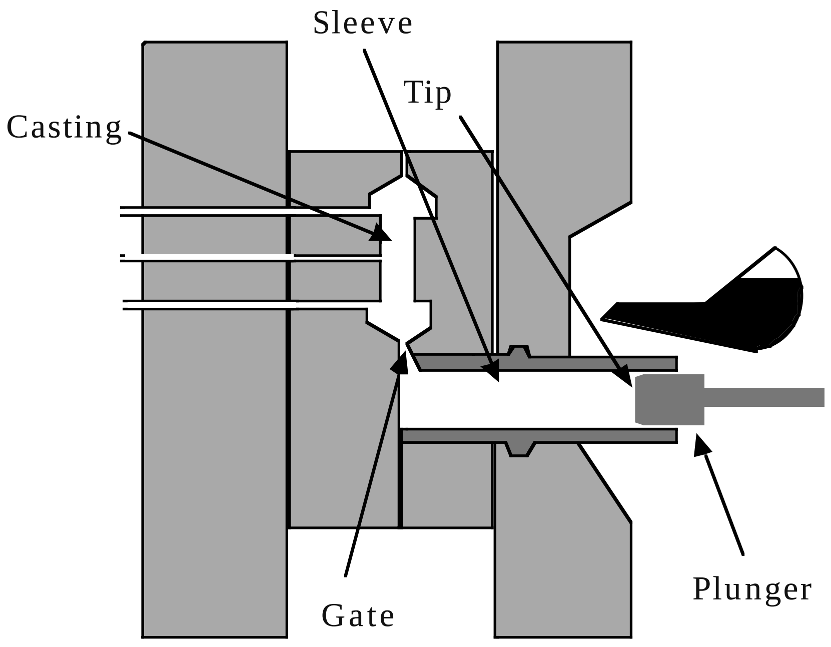


Figure 1: A typical pressure die-casting situation

Machine settings are critical for successful pressure die casting, and will always be a compromise between factors such as cost of producing the casting, maximizing the die life, and quality of the final product.

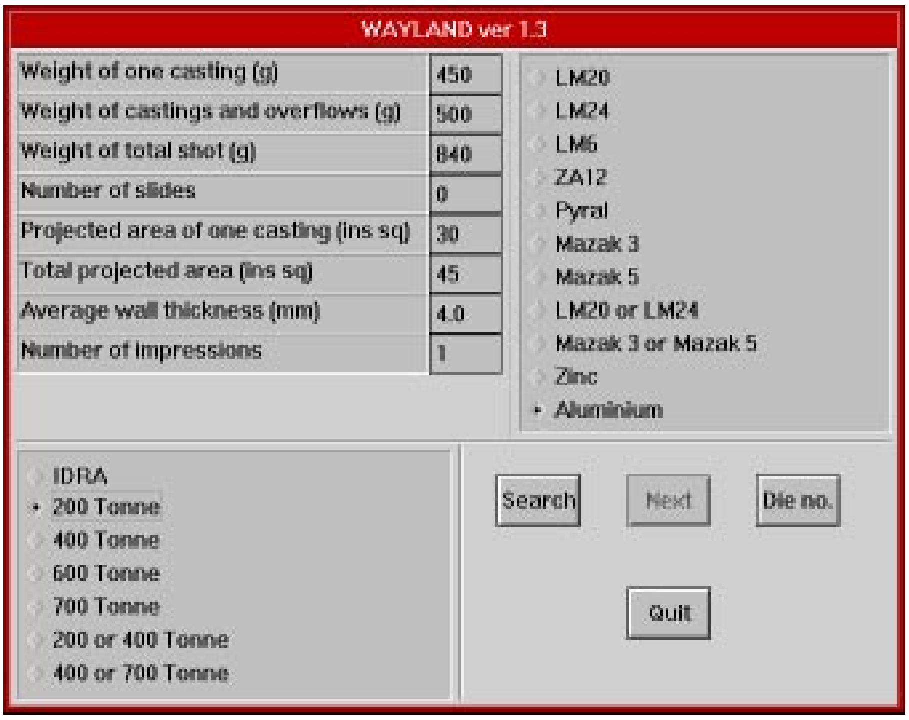
If the machine settings for a new die are badly wrong, then that fact can affect safety. For example, the combination of small tip size, high pressure and small casting can cause the die to “splash”, i.e. the die opens momentarily, spraying molten aluminium at temperatures in excess of 500˚C. The ability to calculate values for machine settings during die design also has commercial implications when bidding for a new casting contract.

There are no generally applicable formulae for calculating values for pressure on metal, gate velocity, cycle time etc. for a given set of conditions, although different engineering bodies have attempted to present formulae to rationalise the process. The different formulae give vastly differing results even for identical operating conditions, so there is no agreed algorithm for calculating results.

The reason for the discrepancies between these formulae is that there are a number of sets of conditions under which a casting can be made. The die parameters are strongly interrelated, making the problem non-decomposable. A change in one parameter can be compensated for by altering another.

Case-based reasoning is an appropriate technology for this problem, because a foundry will tend to have a particular way of working. Foundries which have not built up a body of experience about particular types of casting might use the formulae provided by one of the engineering bodies, but foundries with rather more experience will rely on their own experience. Engineers will refer to records of previous dies with similar input requirements, and adjust the parameters for a similar die to reflect the different requirements of the new die being built. The records of previous dies are good examples of working compromises between the different operating requirements: such compromises might well have been found by costly adjustments performed in the foundry after the die was built.

The Wayland system, described in the following section, automates the identification of past dies with similar characteristics, alters the die settings to take into account the differences between the past die and the new one being designed, and validates that the new solution is within all design limits.


WAYLAND ver 1.3

Weight of one casting (g)	450
Weight of castings and overflows (g)	500
Weight of total shot (g)	840
Number of slides	0
Projected area of one casting (ins sq)	30
Total projected area (ins sq)	45
Average wall thickness (mm)	4.0
Number of impressions	1

LM20
LM24
LM6
ZA12
Pyral
Mazak 3
Mazak 5
LM20 or LM24
Mazak 3 or Mazak 5
Zinc
[✓ Aluminium]

IDRA
[✓ 200 Tonne]
400 Tonne
600 Tonne
700 Tonne
200 or 400 Tonne
400 or 700 Tonne

Figure 2: Wayland input form with values filled in

3. What Wayland does

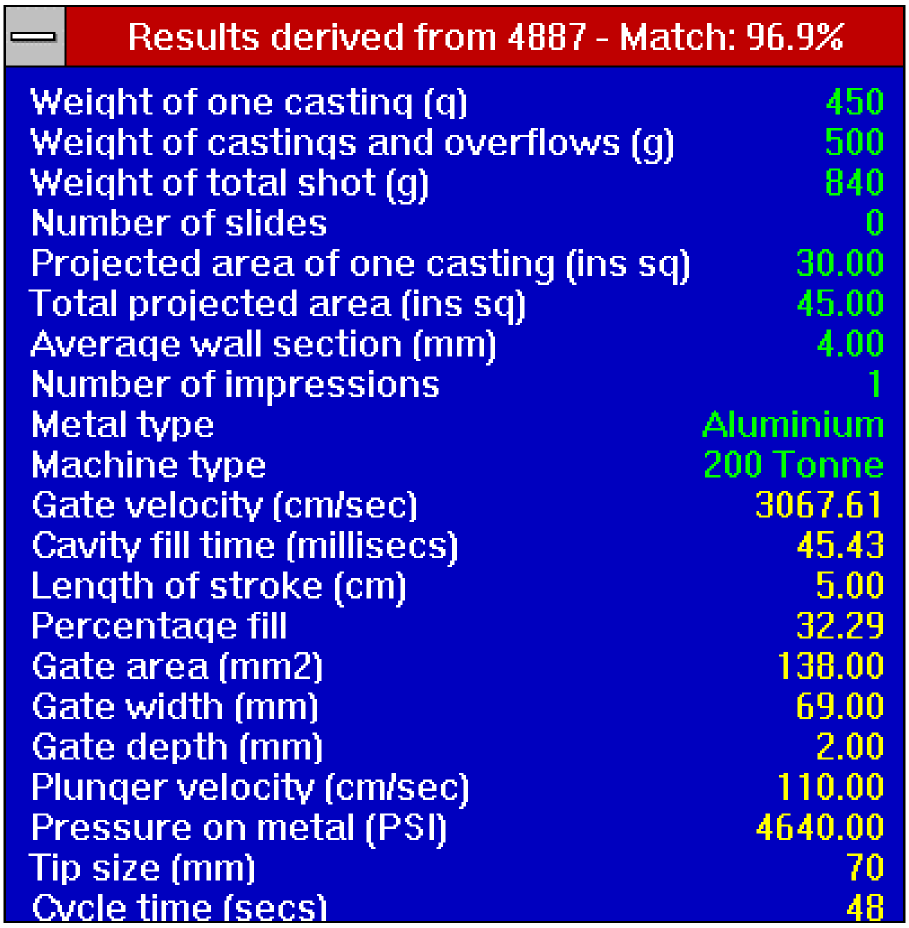
Wayland is available on the foundry computer network, and so users can quickly obtain answers about die design at their own desks. When they start running Wayland, they are presented with a form which they can tab through, filling in the requirements of the new casting: weight, size, number of castings made at one time and the characteristics of the shape. Figure 2 is a screen dump of a filled in form with values specified for each of the input fields.

Clicking on the *Search* button shown in figure 2 will cause Wayland to produce the best matching previous case. This is presented in three windows.

1) Details of the previous case that best matched. This window contains the requirements of the previous case plus the machine setting values being used for that die. This is shown in figure 3.

2) An adapted version of the previous case with the best values for this new die. This window is placed on top of the window with details of the previous case, and slightly to the left of it. This allows the user to compare the two sets of values and to see how the different input requirements have altered the output specification.

3) A picture of the previous casting. The values input in figure 2 do not specify the new die perfectly (they only give an approximation of its shape). The picture allows the user to decide whether the previous case that was selected really is like their new design.


Results derived from 4887 - Match: 96.9%

Weight of one casting (g)	450
Weight of castings and overflows (g)	500
Weight of total shot (g)	840
Number of slides	0
Projected area of one casting (ins sq)	30.00
Total projected area (ins sq)	45.00
Average wall section (mm)	4.00
Number of impressions	1
Metal type	Aluminium
Machine type	200 Tonne
Gate velocity (cm/sec)	3067.61
Cavity fill time (millisecs)	45.43
Length of stroke (cm)	5.00
Percentage fill	32.29
Gate area (mm2)	138.00
Gate width (mm)	69.00
Gate depth (mm)	2.00
Plunger velocity (cm/sec)	110.00
Pressure on metal (PSI)	4640.00
Tip size (mm)	70
Cycle time (secs)	48

Figure 3: Best matching previous case from Wayland

If the user is not happy with the past case that was matched with their problem, then they can press the *Next* button to look at how further cases match their problem. Each time the user presses the *Next* button, the next best matching case will be displayed until the system has shown all matching cases.

4. How Wayland works

4.1 Overview

When the user runs the Wayland program to obtain parameter settings for a new die, the specifications of the die are input as in the example shown in figure 2. The input details are matched against the details of previous dies, and the most appropriate past case is selected from the case base. Finally, rules are then applied to make sure that the best possible solution for the new die is reached.

The rest of this section describes the details of this process, giving examples of cases and rules.

4.2 Finding the best matching case

Wayland has a case base of some 200 previous die designs, extracted from a database of records of actual die performance maintained at the foundry. Only dies with satisfactory performance have their values entered into the case base, so the foundry personnel are confident that each case provides a good basis for calculating new solutions.

Cases are fixed format records, with a field for each of the values shown in figure 3. Some of the fields may be blank, if complete records for a die have not been available. Here is the format of a typical case record.

```
CASE INSTANCE  die_no_5014  IS
        weight_of_casting = 240.00;
        weight_of_casting_and_overflows = 310.00;
        weight_of_total_shot = 520.00;
        no_of_slides = 0.00;
        projected_area_of_casting = 19.50;
        total_projected_area = 35.50;
        average_wall_thickness = 2.75;
        no_of_impressions = 1.00;
        machine_type = t400;
        metal_type = lm24;
SOLUTION IS
        imagefile = 'dn5014.gif';
        gate_velocity = 6414.09;
        cavity_fill_time = 13.77;
        length_of_stroke = 3.10;
        percentage_fill = 16.24;
        gate_area = 135.00;
        gate_width = 90.00;
        gate_depth = 1.50;
        plunger_velocity = 225.00;
        pressure_on_metal = 8000.00;
        tip_size = 70.00;
        cycle_time = 35.00;
END;
```

Preliminary pruning of the case base is done by only retrieving cases for the same type of die-casting machine (e.g. only dies used on the 400 ton machine). Each of the retrieved cases is then assigned an overall match value. This is done by assigning a matching mark to each field and summing the total. Each field is given a weight which expresses its significance (e.g. number of impressions is an important field to match: it specifies how many of the parts are made at once in the die). Matches can be specified as exact (e.g. for number of impressions) or approximate (for items such as weight of casting) where the mark awarded will depend on how close the match is.

The case with the highest overall mark is the best match, and will then have rules applied to it in order to produce the kind of answers shown in figure 3.

4.3 Adapting the best case to fit the new circumstances

Rules are applied to the specification values and to the answers from the past case in order to:

- calculate further information from the specification
- take account of the differences between the past case and the new problem
- change parameters when safety criteria are violated
- decide whether the final result is good enough

This section explains what each kind of rule does, and gives an example of such a rule in Wayland.

Calculating further information from the specification

The simplest type of rule calculates a necessary value directly from the information that the user has given. For example, the total volume of the metal in the casting and overflows is needed in order to calculate other values such as length of plunger stroke. It can be calculated by the following rule:

```
REPAIR RULE find_volume_of_casting_and_overflows IS
        WHEN volume_of_casting_and_overflows IS UNDEFINED
    THEN
        EVALUATE volume_of_casting_and_overflows TO
            weight_of_casting_and_overflows / 0.0026;
    END;
```

Adapting values from the past case for the new die

This type of rule takes results from the past case and adapts them to the differing circumstances of the new die. For example, the following rule uses the length of the plunger stroke for filling the new die (calculated by the first kind of rule) and the value for plunger velocity taken from the past case, and calculates the time it takes to push the metal into the die.

```
REPAIR RULE find_cavity_fill_time IS
    WHEN cavity_fill_time IS UNDEFINED
THEN
    EVALUATE cavity_fill_time TO
        (length_of_stroke / plunger_velocity) * 1000;
END;
```

Changing parameters when safety criteria are violated

Some of the rules check whether the operating values are within safe boundaries and change them if they are not. The following example checks that the velocity of the metal through the gate is not too high. If it is, then it increases the size of the gate, so that the velocity will be reduced. The command “REPAIR” states that all rules should then be re-evaluated. This enables the rule to fire more than once if the gate velocity is still too high.

```
REPAIR RULE gate_velocity_too_big IS
        WHEN gate_velocity >= 5000
THEN
        EVALUATE gate_area TO gate_area + 10;
        EVALUATE gate_depth TO (gate_area / gate_width) / no_of_impressions;
        pr(['Warning: changed gate area: gate velocity too big']);
        pr(['Warning: changed gate depth: gate velocity too big']);
        REPAIR;
END;
```

Deciding whether the final solution is safe

Some problems caused by the difference between the past case and the new die are too complex for the Wayland program to deal with in the way shown above for gate velocity. In those cases, the user is warned about the problem. The following example warns that the job is too large to run on any of the machines in the foundry.

```
REPAIR RULE no_machine_big_enough IS
        WHEN total_projected_area >= 160
THEN
        pr(['Warning: projected area too big for 700 tonne machine.’]);
END;
```

5. Benefits of Wayland

Wayland has been available at the foundry for more than two years, and is used by several different kinds of foundry personnel. They are interested in different information from the case base. For some users, the information that they actually want is not held on-line in the case base, but the best matching case can be used to index further information that is held not on computer but in manual files. Such users still find this much faster and more effective than browsing through paper records of past dies to find relevant information.

The main types of user can be categorized as:

- Sales staff
- Die designers
- Foundry engineers

5.1 Sales staff

Wayland is in daily use to provide estimates of the cost of producing a new component. Sales staff are most interested in cycle time, although close matches in the case base are also used to access off-line information such as the cost of building the die to manufacturing such a component.

Before Wayland was available, sales staff had either to obtain an accurate estimate from an experienced engineer (taking several hours), or give a 'finger in the air' estimate: very unsatisfactory in the highly competitive foundry industry.

5.2 Die designers

A close match with an existing die design can be helpful when designing a new die. It can enable the designer to reuse an existing design for 'running and gating' (the layout for metal feeding into and overflowing from a die). This can save several days work. More importantly, it bases the new design on a previous design known to be successful, and so minimises the risk that the new design will not work properly, or will have a short lifespan.

5.3 Foundry engineers

This is the area where Wayland was originally intended to be used. By providing the most accurate available values for parameter settings, it can save significant setup time when a new die is first being installed. Previously, several days might have been spent experimenting with settings such as plunger velocity, in order to produce acceptable castings. In some cases, the physical shape of the die needed to be altered in order to produce acceptable results, causing further expense and loss of production. The use of Wayland has significantly reduced this time.

The Wayland case base is maintained by one of the foundry engineers, and perhaps the best measure of its success is that he is happy to have the task of maintenance. The amount of time it saves him and his colleagues far outweighs the effort of occasionally adding further cases.

The engineers are imaginative in their use of Wayland and produce new ways in which it can help them. One of the most recent innovations was using Wayland in troubleshooting. The engineers had had problems with one particular die for some time, and had been unable to make it work well consistently. An engineer decided to enter its parameters to Wayland. There was a good match on an existing case, and Wayland recommended a larger gate area than there was on the problem die. The gate on the problem die was altered accordingly, and the problems went away.

5.4 Summary of benefits

The benefits of Wayland can be broken into three main areas:

It replaces opinion with reference to actual experience. In the troubleshooting example just quoted, different engineers had different opinions of how to fix the problems. Wayland referred to a good previous solution and used that as a basis for its recommendations.

It saves engineer time. Much less time needs to be spent altering dies or changing parameters.

It reduces scrap. Less bad castings are made, because the parameters are correct.

It produces accurate estimates. This could be done before, but only by expending engineer time on a very speculative exercise. The foundry can produce much more competitive tenders in a fiercely competitive market.

Perhaps the most telling item in favour of the system us that the benefits clear enough to decide to use Wayland elsewhere. The foundry where the system has been deployed is part of a group of three foundries, and the system is also being deployed in the other two foundries in the group. This will be done using their own case bases, as their methods of working and typical dies are different.

6. Lessons for this type of system

This section draws together the lessons that have been learned from the development of Wayland, and discusses how generally applicable they are.

6.1 Characteristics of the problem

Wayland has a database of input values and corresponding output values, where the relationship between the two has been obtained from past experience of correct solutions to similar problems. The further characteristics that made case-based reasoning a good solution to the problem were:

Large value space for inputs and outputs. Many of the input variables are continuous. and can vary quite widely, giving an infinite combination of possible sets of inputs.

Relationship between inputs and outputs unclear. There are no complete algorithms to give reliable output values for all inputs. The multi-dimensional nature of the problem would make it very difficult to discover such an algorithm.

Local adjustment of parameters provides reasonable results. While there are no global algorithms for providing correct answers, it is possible to adjust an answer locally so that the new solution works correctly for the new problem.

A completely accurate solution is not needed. There is a space of correct solutions, where the system will work adequately.

Wayland is effective because its case base provides local reference points in the global space of all possible casting problems. The repair rules then provide both local adjustment, and verification checks that the new answer does not violate any global constraints.

The task which Wayland carries out can be identified in other industries as well. Obvious examples are in related industries. For example, we are constructing a system for a similar problem in a plastic injection moulding company. However, in order to illustrate that such problems are common throughout engineering, we will give two further examples:

Design of standard highway bridges. Highway bridges are the most common type of bridge built in modern times, and large construction companies have designs for many such bridges. A bridge design only tends to be reused where an engineer has specific experience of a previous similar design. We have constructed a prototype working in a similar way to Wayland, which accesses previous designs with similar characteristics [Moore et al.]. This would enable many of the same features that Wayland does (accurate estimating, method of construction for previous design).

Deciding on mix when making steel. When scrap steel is added to a blast furnace, it is necessary to decide how much lime and iron to mix together in order to make the right quality of steel. The answer will depend on the type and amount of scrap being added. Deciding on the correct mix seems to be an arcane skill, where previous experience counts for much. Such experience is available in abundance in the form of previous records of mixes and whether they were successful or not. Again, minor repairs to the closest solution would give a fairly accurate answer.

6.2 Case-based reasoning is an effective solution

It is reasonable to consider whether other technologies might not provide a better solution to the kinds of problem just described.

The minimalist approach

Does this problem need case-based reasoning? Could it not be solved by putting past cases in a spreadsheet or database and writing some macros or C code to access relevant cases.

The answer is undoubtedly "yes, it could be solved with more conventional technology". The advantage of CBR is that it makes the knowledge explicit. Both the matching rules and the repair rules are written in an understandable and maintainable way.

The subsymbolic approach

Why not use a neural net for this type of problem? Nets are excellent at learning to make associations.

It is not clear what examples could be given to a net to train it. A number of past examples of good solutions exist, but no examples of what is a good match. In effect, the person constructing the net would be generating weights for fields in order to produce examples to give to the net, so that it could produce an overall matching expression. It seems easier to keep the allocation of weights explicit, so that the system is more accountable.

The symbolic approach

Something like this could be built in a knowledge representation language.

Indeed it could. Each case would be an object of class "case type", and you could write a number of rules to do the matching and adaptation of cases. This would provide a reasonable solution. The argument against doing this is the same as the argument against using a spreadsheet. Case-based reasoning provides a more specific paradigm for building such systems. The case-based reasoning concepts of having a case, performing matching on the case and then adapting the resultant solution are identifiable in this type of application. It is reasonable to use a tool which explicitly supports those concepts.

6.3 Design lessons

Given that such tasks exist in different branches of engineering, and that case-based reasoning with repair rules is an effective solution for such applications, this subsection discusses issues that arise when building such an application.

Provide flexibility

Wayland was originally designed for one type of user, foundry engineers. Early versions of the system expected users to enter values for all input fields and to take the best result provided and use it.

In fact, users have different ways of working. Some do work in the way described (salesmen, for instance). Engineers prefer to browse through several of the best matches looking at the information, so we provided the *Next* button for them. Both foundry engineers and die designers like the ability to look at all examples of solutions to a particular kind of problem – all three impression dies with two sliders, for example – so it is important that they should be able to do this more traditional database retrieval as well.

Provide the maximum amount of information

Such systems are of maximum benefit when users can access all of the information related to the case. Technical users will want to see the original case as well as the set of adapted answers that are the new solution. They may want details of the original case which are not part of the adapted answer. Cost of producing a die is an example of that in Wayland. If that information is not in the case itself, it is important that the case at least provides a pointer to where it can be found.

Provide guidelines for interpreting answers

Technical users will have their own opinions as to whether an answer from the system is valid. For salesmen and other less technical users, it is important to provide guidelines on how the system should be used. The match has a weight attached to it: what values are good matches? Warnings are provided when altering significant values in previous cases: when should the user decide that the warnings mean that the case is not valid. We have provided a set of written guidelines for the users on these matters. In hindsight, it would have been better to provide further repair rules which did this interpretation automatically.

Acknowledgements

This work was carried out at Kaye (Presteigne) Ltd on SERC TCS grant GR/H50043.

References

D. Hinkle and C Toomey, Clavier: applying case-based reasoning to composite part fabrication, Proceedings 6th Innovative Applications of Artificial Intelligence Conference, Seattle, Washington, pp55–62, AAAI Press, 1994.

D. Hennessy, D. Hinkle, Applying case-based reasoning to autoclave loading, IEEE Expert 7(5), pp21–26, 1992.

J. L. Kolodner, Case-based reasoning, Morgan Kaufmann, 1993.

C. J. Moore, M. Lehane, C. J. Price, Procs IEE Colloquium on Case Based Reasoning, London, February 1993.

C. J. Price, I. S. Pegler, F. Bell, *Case-based reasoning in the melting pot*, International Journal of Applied Expert Systems, volume 1(2), 1993.