13

Object Oriented information storage for the design of injection moulds

R. Willems, D. Lecluse, J.P. Kruth
Dept. of Mech. Eng., Division PMA, K.U.Leuven
Celestijnenlaan 300B B-3001 Heverlee Belgium
Tel.: (+32)16 32 24 80, Fax.: (+32)16 32 29 87

Abstract

This paper describes the various levels of information within an intelligent support system for the design of injection moulds. The design system integrates CAD/CAM, database and expert system tools. It allows the designer to define an injection mould in terms of high level objects and features that match his conceptual way of thinking. The data of standard mould components, that are available from a number of suppliers, is stored in a relational database. The generated information on geometry, technology and functionality is stored in a feature based, object oriented model, the so called mould model. The design expertise, that is of a more changing nature, is represented in IF-THEN type rules which are gathered in different knowledge sources.

Keywords

CAD/CAM, Injection mould design, Object Orientation, Rule Based system

1 INTRODUCTION

The efficiency of commercial CAD/CAM systems depends upon the existence of modules that are tailored to the specific needs of the user. Because the design of injection moulds is a complex process that requires a lot of experience, specific CAD/CAM software to support this design process is not widely available. 'Mould libraries' that support the designer with the selection and drawing of the various standard components contributed to a more effective use of CAD/CAM systems in mould making industry (Kruth and Kesteloot, 1989; Schwarz, 1985). However, design tasks related to the selection of standard components require only very little expertise. The main difficulties are encountered in the design of so called "forming parts", such as inserts and slides which are not standard because they are closely related to the particular shape of the plastic product to be moulded. Moreover, present day systems usually provide only two-dimensional engineering drawings, representing the mould in terms of low level entities such as points, lines and circles. This type of representation is not convenient for further support during the design and manufacturing process of injection moulds.

T. Tomiyama et al. (eds.), *Knowledge Intensive CAD*

In the framework of the BRITE-EURAM project PROMISES (project no. 3148) a prototype of an intelligent support system for the design of injection moulds has been developed (Kruth and Willems, 1994). This system, integrating CAD/CAM, database and expert system tools (Figure 1), allows the designer to define an injection mould in terms of high level objects and features that match his conceptional way of thinking. To achieve a straightforward integration with various existing CAD/CAM systems a neutral CAD programming interface was used. This interface, developed under the BRITE-project MODESTI (project no. 1391), comprises a number of standardised C-routines. By implementing this limited amount of BM-functions (BM stands for BRITE-MODESTI) the application software becomes portable to another CAD-system.

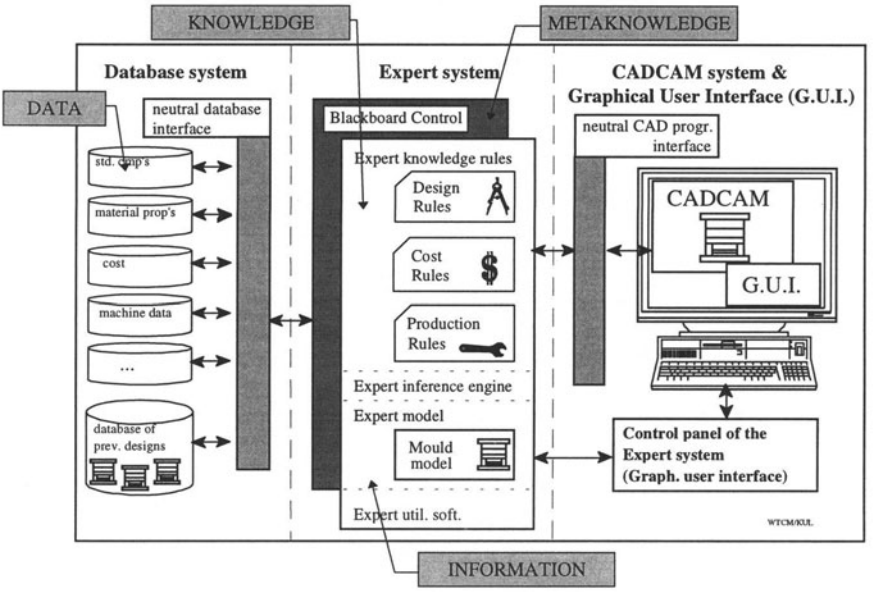

Figure 1 The integration of CAD/CAM, database and expert system tools.

2 LEVELS OF INFORMATION AND KNOWLEDGE

Intelligence has been usefully described as an amalgam of many information-representation and information-processing talents and this certainly applies to computer intelligence (Winston, 1984). The way the information is represented and stored, clearly has a great effect on the convenience of making computers acting like intelligent design support tools. As used in this article, information is only one part of a more general knowledge hierarchy that has been described by Giarratano and Riley (1989). Data and information constitute the lower levels in this general knowledge pyramid, having expertise as the next higher level and metaknowledge on top of the hierarchy. Data is defined as the whole of items of potential interest, while information is processed data of particular interest. Metaknowledge is knowledge about knowledge.

This article describes the various levels of information within the PROMISES design support system. The next section shortly discusses the database of standard components, storing the whole of items of potential interest to any mould design. Section 4 describes more thoroughly the mould model, storing all processed data of particular interest for a specific mould design. Finally, section 5 demonstrates the representation of knowledge or expertise in rules. Since the prototype software relies on a simplified blackboard concept for the management of the various knowledge sources, the system even incorporates a kind of rudimentary metaknowledge.

3 DATA STORAGE: STANDARD COMPONENT DATABASE

Most moulds today are built up of standard components such as plates, screws, bushes, etc. which allows the designer to concentrate on the unique elements in the mould. These standard mould components are available from a number of suppliers, like HASCO, DME and EOC to name only a few. Information about these standard components is clearly of potential interest for any injection mould design. The designer must be able to access the contents of these catalogues so as to insert components into the mould. The data for these catalogues is stored in a relational database that is laid out as a series of tables. The amount of data stored is minimised by implementing a set of tables for each component type, one of which contains the fixed data and the other containing the selectable data. The database is accessed with the standard SQL query language which ensures that the software can work with different commercial database systems.

4 INFORMATION STORAGE: MOULD MODEL

Conventional CAD systems are not giving sufficient support to the designer because they mainly work with primitive geometric objects such as lines, points and surfaces, while a designer mostly thinks and works with high level design-oriented objects. Moreover, it has been realised that traditional CAD databases are not the best vehicle to provide sufficient information to the downstream applications of CAPP and CAM (Kusiak et *al.*, 1991; Wierda, 1991). Therefore we developed a secondary representation to hold high level design information outside the CAD system, independent from the data structure of the used CAD system. While the user designs a mould, all the generated information on geometry, technology and functionality is stored in a feature based, object oriented model, the so called *mould model* (Figure 1 and Figure 2). As a result, the prototype system supports two models in parallel: one being the CAD representation in terms of low level graphic entities, the other being the mould model.

4.1 Object oriented

While this is not the right place to contribute to the discussion of the object oriented paradigm, we give just a short outline of the basic concepts of object orientation. Object orientation is based on the *object*-concept. An object can be defined as any collection of abstract data. However, an object not only consists of a set of data or attributes, it also

incorporates a set of methods which can operate on that data. This integration of data and methods is called *encapsulation.* Encapsulation is a specific form of information hiding, namely the hiding of a data structure and the implementation of its associated operations within the same module.

Objects can represent real world objects such as plates, screws and ejector pins as well as more abstract entities such as processes, organisations and conditions. In object oriented analysis the world is considered as a number of objects and a first step is the identification of the interrelated classes of objects in the application area. Classes constitute groups of objects that are all characterised by common properties and behaviour and so share the same set of attributes and methods. It is important to see the difference between a class and an instance of that class; an instance is a particular entity whereas the class is an abstraction. A basic point to mention also is that there is an important difference in using an object oriented programming language and using an object oriented programming style. If the term object oriented language means anything it must mean a programming language that provides mechanisms that support the object oriented style of programming (applying information hiding, data abstraction, inheritance, generic classes, etc.) (Meyer, 1988).

The advantages of object oriented programming are well documented and include reusability, extensibility and fast prototyping, (Stroustrup, 1988). Moreover, object oriented programming is argued to be well suited for engineering applications such as CAD/CAM, because the design methodology and the think process to develop such programs are almost identical to those that users follow to analyse the application (Zeid, 1991).

The mould model is implemented in LISP-FLAVORS. LISP is a well known AI-language but, as such, is not an object oriented language. FLAVORS (a predecessor to CLOS) is an object oriented shell on top of LISP (comparable with other object oriented extensions like C++, Object PASCAL, etc.).

4.2 Object classification and taxonomy

As stated before, object oriented analysis tries to identify interrelated classes of objects. The classification of objects results in a hierarchical class structure with base classes and subclasses (Figure 2). A subclass is a specialisation of its base class and inherits all the attributes and methods. However, subclasses may have overriding or additional methods and attributes of their own. A clamping plate, for instance, is a special kind of plate with characteristics and behaviour similar to any arbitrary plate in the world (length, width and thickness) but with some additional features because of its specific function (clamping purpose) in a mould assembly.

Figure 2 shows that the base class of the mould model is the MOULD_OBJECT class. This class incorporates any collection of abstract data that is relevant for the design of an injection mould. Since a mould basically is a 3D assembly of components and features, any MOULD_OBJECT is considered to be a MOULD_COMPONENT or a MOULD_FEATURE. The MOULD_COMPONENT class represents all the physical objects in a mould such as plates, screws, slides, inserts, ejector pins, etc. The MOULD_FEATURE class represents all the other mould objects. A hole is a typical example of a feature being related to another component; a hole does not exist on its own but is always 'made in' another physical component.

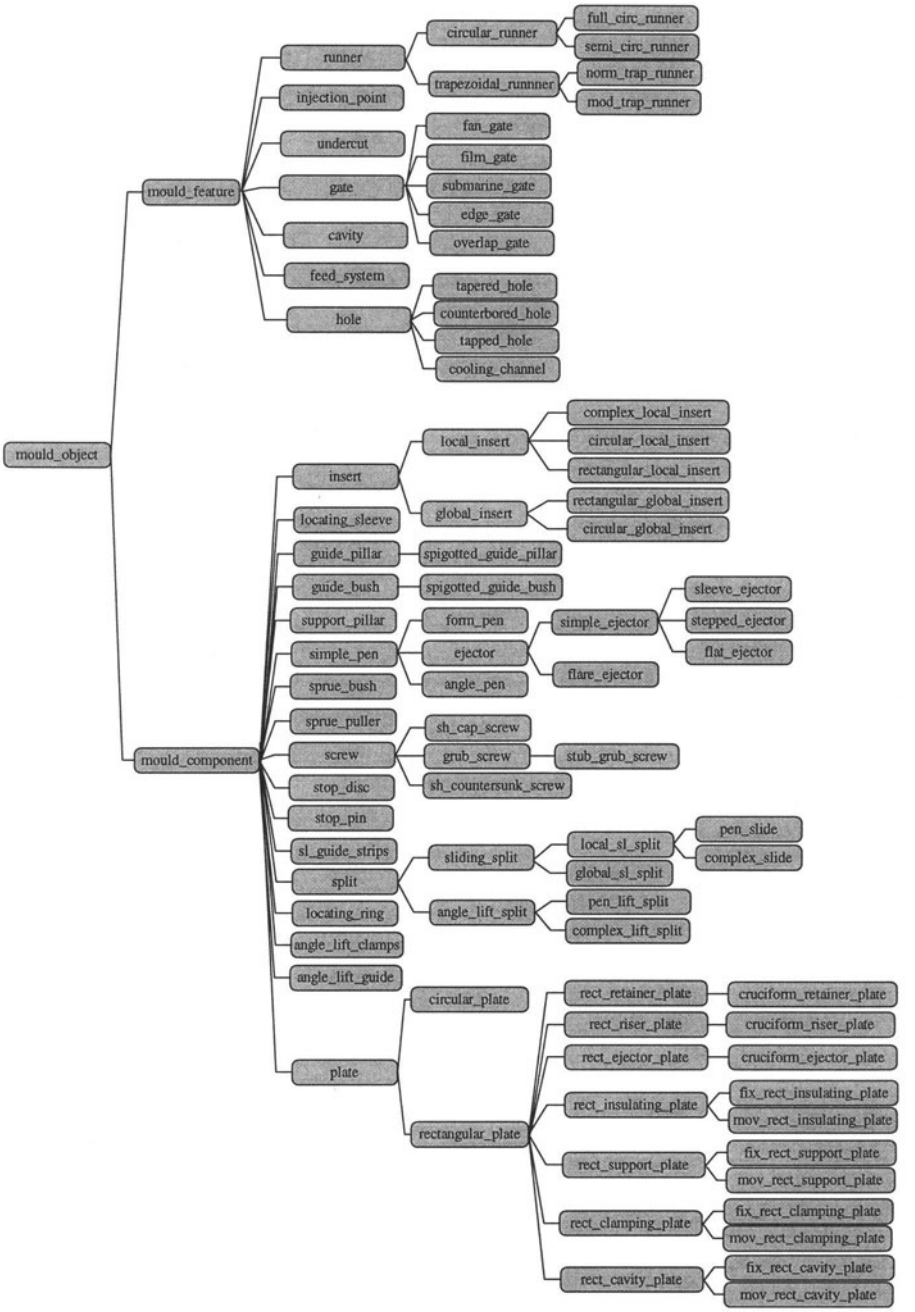

Figure 2 Overview of the different classes in the mould model.

The feature concept originally stems from the research in process planning but was later extended and introduced in other engineering areas (Salomons et *al.,* 1992). Shah (1990) defines a feature as any collection of abstract data that aid the design or the communication between the design and manufacturing or any other engineering application. The attentive reader will recognise the analogy of this definition with the previously stated definition of an object. Indeed, the feature based approach may be considered as a logical consequence of the object oriented approach or vice versa. Subscribing the general feature definition of Shah, the MOULD_FEATURE class not only contains the traditional manufacturing features used in process planning (e.g. holes) but also more abstract design features which are of special interest for injection mould design (e.g. undercuts, injection points, cooling channels and runners). Mould features match the level of abstraction on which engineers think, they are objects in engineering reasoning processes (Wierda, 1991).

The MOULD_OBJECT class contains the following attributes:

- *cost*: stores the cost of the object;
- *pos*: position is stored as a list of three coordinates (x, y, z);
- *dir*: the orientation is stored as a list of three rotation angles (rx, ry, rz);
- *name*: name of the object;
- *parent*: object to which its position is related;
- *children*: list of objects that are positioned relative to the object;
- *master*: object that is the master of this object;
- *subordinates*: list of objects that are subordinate to this object;
- *standard*: flag indicating whether this object is standard or not (i.e. whether it can be bought from a standard component supplier or not).

Besides the common attributes of all objects in a mould, the MOULD_OBJECT class incorporates the basic functionality (by means of class member functions or methods) of all mould objects. Most of these basic functions deal with assembly management: *find-position-in-mould*, *find-position-relative-to*, *find-local-position*, *set-pos*, *set-dir*. The meaning of which is clarified in section 4.4.

The MOULD_COMPONENT class contains the following attributes:

- *manufacturer*: name of supplier (if component is purchased from a standard component supplier);
- *BOM-code*: bill of material code;
- *material*: material code.

Being the base class of all the physical objects in a mould, the MOULD_COMPONENT class is also the base class of all the components that may be purchased from standard component suppliers. Therefore, this class incorporates the general functions for accessing the data of the standard catalogues. However, the *standard* attribute is not only applicable to mould components but to all mould objects. The holes that are present in the standard mould plates are stored in separate mould features that are marked as standard, in order to prevent them from being changed by the designer.

All objects within the mould model contain a certain amount of geometrical information that is of a higher level then the one that is commonly stored in the CAD database. The geometrical information for plates for instance is limited to three parameters: length, width and thickness (Figure 3). The information of the holes is stored separately but the holes are related to the plate by parent-child relationships (see section 4.3). Each hole has an attribute *purpose,* indicating the purpose of the hole, e.g. a counterbored hole could be for a screw head or for a guide pillar or guide bush. In each of these cases the object is of the same class, but it performs a different function and so is likely to be machined differently. It may seem that a subclass is required for this, but then a lot of such subclasses would be required.

Since most of the moulds are built up of similar components, the geometrical parameters to be stored greatly resemble the various parameters that are also found in the standard catalogues from HASCO, EOC, DME, etc. The class hierarchy model itself does not explicitly draws a distinction between standard and nonstandard components, in the sense that they do not constitute two separate classes. In contrast each object has an attribute *standard.* The essential idea is that, when a designer wishes to insert a component, he could choose data for that component by selecting options from the suppliers catalogue or he could enter this data himself. The mould design software takes this data, not really caring where it originated, and creates an object of the appropriate class storing the corresponding information in the instance's attributes. When the data is chosen from a standard catalogue the *standard* attribute is set true. Thereafter, the behaviour of the mould component is controlled by the definition of the class of which it is an instance. It is worth to mention that even when the dimensions and holes are taken from the standard component database the component may be indicated as nonstandard afterwards, because, for a number of reasons, a designer may decide to manufacture the component himself.

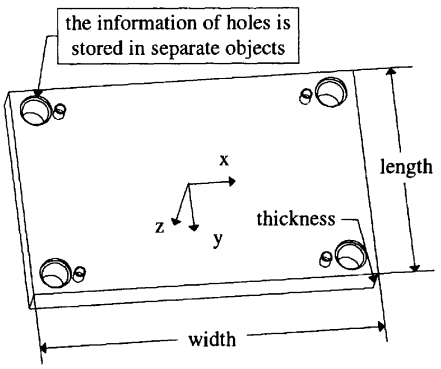

Figure 3 Geometrical attributes of a plate.

Besides the commonly used standard components, there are a number of nonstandard components mostly located in the vicinity of the cavity. These components are strongly related to the particular shape of the product and are referred to as forming parts; e.g. slides and inserts. The geometrical attributes of these forming parts are limited to the product independent shape of it (Figure 4).

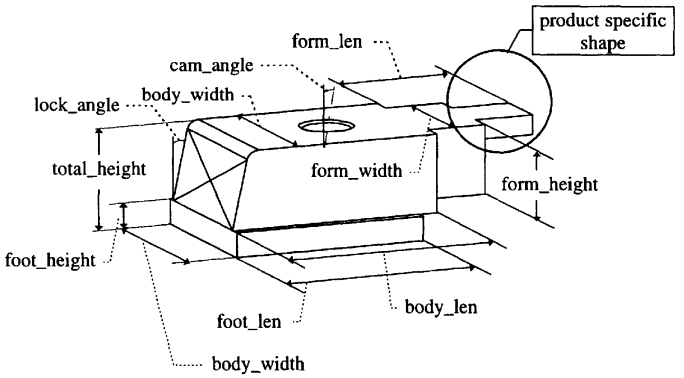

Figure 4 Geometrical attributes of a complex sliding split.

4.3 Assembly representation

We have previously stated that a mould is a topological 3D assembly of components and features. The configuration of these components in space is important design information that must be stored. The simplest way to represent an assembly is by specifying the location and orientation of each object with respect to the global coordinate system of the assembly. The location and orientation can be specified by means of a 4x4 homogeneous transformation matrix that transforms the local coordinates of the geometric entities of the part to the global coordinate system of the assembly:

$$X_m = [T]X_{comp} \tag{1}$$

X_{comp} is the position of a point relative to the local components coordinate system and X_m is the position of the point relative to the mould coordinate system. Each vector is given by:

$$X = [x \; y \; z \; 1]^T \tag{2}$$

The matrix T is the homogeneous transformation matrix. It is a 4x4 matrix given by:

$$[T] = \left[\begin{array}{ccc|c} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R & P \\ \hline 0\,0\,0 & 1 \end{array}\right] \tag{3}$$

R is the rotation matrix that defines the orientation of the local coordinate system to the assembly coordinate system. P is the position vector that describes the origin of the local coordinate system relative to the assembly coordinate system. The columns of R are given by the direction cosines of the unit vectors of the local coordinate system relative to the mould coordinate system.

Rather than storing the entire transformation matrix it is preferable to store the position and direction of each object separately. The position of an arbitrary object in space is fully determined by six parameters: xc, yc, zc, rx, ry, rz (six degrees of freedom). Therefore we store the position as an attribute *pos* (xc yc zc), i.e. a list of the three linear degrees of freedom and the orientation as an attribute *dir* (rx ry rz), i.e. a list of the three rotational degrees of freedom.

Location of an object is a relative property. This implies that the position of one object is always related to the coordinate system of another (reference) object. The mould, having its own coordinate system, is the basic reference for all the components in the mould assembly. However, it is not always useful to refer a component's position directly to the mould assembly. A hole for instance is likely to be positioned relative to a plate while the plate on its turn is positioned relative to the assembly. In this case, successive transformations will yield a component's absolute position in the mould coordinate system. The reference information is indicated by a parent-child relation, e.g. when the position of a hole is relative to a plate; the plate is the parent of the hole, the hole is a child of the plate. Working with relative positioning is very easy when some displacements have to be done. The translation of a plate for instance, requires the position to be changed but since all the holes in the plate are positioned relative to the plate, these holes are automatically displaced without changing any of their (relative) positions.

Because of the similarity in the structural built up of injection moulds, considerable 'intelligence' is implemented on the level of this assembly management. The system is able to position a lot of components automatically, based on their functionality and their relation to other components. A slide, for instance, is able to infer its position out of the position of the related undercuts. However, the system always allows the user to adapt the parent-child relations and the position of every single object.

4.4 Relations between components and features

In section 4.2 it was shown that the mould model contains high level geometrical information. It is known that an approach based on geometrical information only, does not satisfy the demands posed to a modern design support system. Therefore the mould model also stores a great deal of technological information such as material codes, BOM codes, costs, suppliers, etc. Moreover, it stores a number of different relationships between the various objects (components and features) in a mould.

Parent-child relation

As discussed in previous section, the parent-child relation is important for the relative positioning of the different objects in the mould assembly. The parent-child relation is a 'one-to-many' relation; each object can only have one parent but every parent can have different children. The various parent-child relations provide a kind of topological tree (or location graph) of the mould. This topological tree is not to be mixed up with the class hierarchical model. Figure 5 shows the default parent-child relations that are established by the system between objects of the various classes. Because positions may be referring to different objects, it is essential to compare only those positions that are related to the same reference coordinate system. Therefore, it is sometimes required to transform one of the positions to another reference by means of successive coordinate transformations. The basic functions for

searching in the location graph and applying the corresponding coordinate transformations are incorporated in the MOULD_OBJECT class (find-position-in-mould, find-position-relative-to, find-local-position, find-all-children).

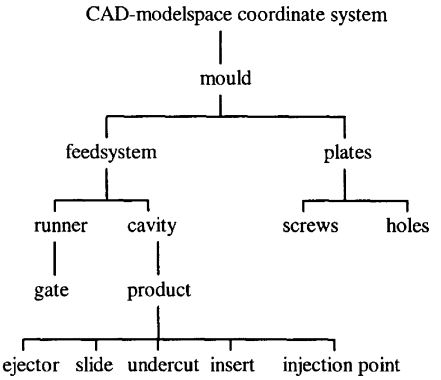

Figure 5 Topological tree of a mould.

Master-subordinate relation

The master-subordinate relation indicates which components are derived from each other and must be manipulated together, e.g. a hole for an ejector pin is a subordinate of this ejector pin and the ejector pin is the master of the hole (Figure 6). An important consequence of this relation is that the hole will be deleted when the ejector pin is deleted, because the ejector pin was the only reason of existence for the hole. Notice that a subordinate is not always positioned relative to its master (but it is possible). In Figure 6, for instance, the hole in the support plate is positioned relative to the plate but is subordinate to the ejector pin. The master-subordinate relation allows the system to change the position of the ejector pin hole whenever the position of the ejector pin is changed. The master-subordinate relation is also a 'one-to-many' relation.

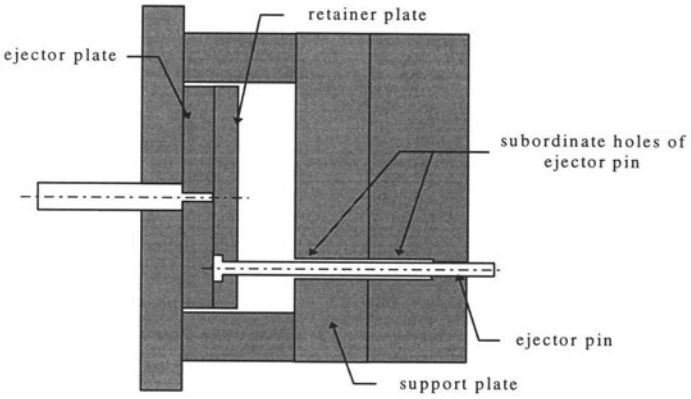

Figure 6 Master-subordinate relation between ejector pin and hole.

Undercut-slide relation

The starting point for the design of slides are undercuts. Undercuts are considered as special objects (features) that are related to the product but can be managed separately. In literature different types of undercuts are distinguished: external, internal and threaded undercuts or local and global undercuts (Pye, 1989; Menges and Mohren, 1986). Within the mould model this distinction is not made because terms like external, internal, local and global are not always well defined (see Figure 7). Even when you can define unambiguously whether an undercut is local, global, internal or external, it will not necessarily result in a predefined demoulding solution. For this reason, the design support system leaves the selection of the appropriate demoulding solution to the designer. An experienced mould designer will know the type of slide to be used at once, but still has to spend considerable effort on calculating the appropriate dimensions, the suitable angle for the actuating finger cam and designing guide strips, locking heel, etc. The slide objects in the mould model have the intelligence to provide these computations and to generate an appropriate representation on the CAD system. Because of the slide-undercut relation, modifications to undercuts allow an automatic update of the corresponding slides (Kruth and Willems, 1994).

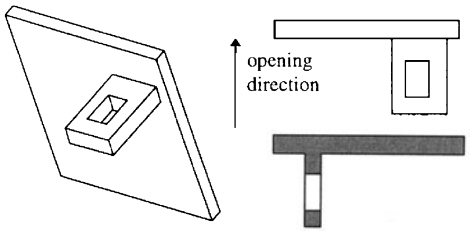

Figure 7 Internal or external undercut?

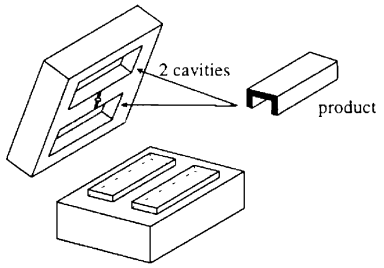

Figure 8 One product and two cavities in a two-impression mould.

Cavity-product relation

Most of the injection moulds form one particular product but contain several impressions (the multi-impression moulds), each impression producing the same product. In order to make a clear distinction between the product to be moulded and the cavity impressions that are forming this product, there is a class named CAVITY. Having cavities as separate objects, makes it possible to move, to delete or to perform any other operation on the cavity impressions without disturbing the moulded product. The product is considered as a general object, containing a lot of input information that applies to all the individual cavities. Most moulds will only have one product involved, though it is possible to have several different products in one mould. In this case some cavities are linked with one product, while others are related to another product.

5 EXPERTISE STORAGE: RULES

It is mentioned elsewhere that the PROMISES prototype system integrates relational database, CAD/CAM and expert system tools. Expert systems are usually described as computer programs that emulate the decision making ability of human experts (Coyne *et al.*, 1990). Successful expert systems today are mostly knowledge based, a concept that was primarily introduced with the famous expert system MYCIN. The key to such knowledge-based systems lies in the clear separation between knowledge and reasoning. Reasoning can be described as the processing or control of knowledge and is typically a human activity. The term inferencing is generally used for 'mechanical systems', and the mechanism that is performing the inferencing is generally called an inference engine. With LISP as development language, an inference mechanism and a knowledge representation suitable to the project was developed. LISP does not offer any inference mechanism or knowledge representation. Two strategies of knowledge processing were implemented in the inference engine: backward chaining and forward chaining. In backward chaining, it is tried to establish whether a fact is true by chaining back through all the rules until we arrive at facts that are known to be true. The multiple inferences that connect a goal with its preconditions, i.e. a chain is traversed from the goal (or conclusion) back to the required facts (or preconditions). A chain that is traversed from facts to conclusions is called a forward chain. The backward reasoning is basically invoked to define unknown elements while the forward chaining is mainly used for verification and modification purposes.

Like most commercial expert systems, PROMISES represents the knowledge by means of traditional IF-THEN type rules. The syntax of the rules is designed so that all the power of LISP can be used within the rules; for example: functions can be called within the rules. The possibility to call LISP expressions from within the rules, allows to have a strong combination of procedural and rule based reasoning mechanisms.
The syntax of a rule is the following:

(**defrule** <rule name>

: IF (**known** <data_element> *)

<IF-TEST : LISP-expression>

: THEN (**defines** <data_element> *)

<THEN-action : LISP-expression>

)

A data element can be an instance name, an instance attribute or an arbitrary LISP variable. In practice a lot of rules apply to a group of objects and it would be unacceptable to write the same rule for each of the different object names. Therefore, we provided the possibility to have data elements like class patterns. A pattern is a description of a well-defined group of objects. Within the PROMISES system, a pattern is either the description of a class (giving access to the subclasses), or the description of an attribute of a class (giving also access to subclasses). This ability of 'pattern matching' allows that rules are written on the level of, for instance, an ejector pin class, hereby applying to all instances of that class.

The rules are organised in sets, called knowledge sources. These knowledge sources act as real world experts gathered in front of a blackboard and solving the problem. Each expert has his own experience, solves part of the problem and puts his findings on the blackboard. The user of the application is considered as a special knowledge source that is asked when no other knowledge source can contribute to the solution of the problem. The blackboard control acts like a moderator, selecting appropriate knowledge sources in an opportunistic way. The blackboard control incorporates a kind of rudimentary metaknowledge (Lecluse and Sleeckx, 1993). Metaknowledge is said to be the top level of the general knowledge pyramid.

The PROMISES design support system contains rulesets for product definition, cavity layout, cooling, demoulding mechanisms, inserts and cost estimation. Since the blackboard controller ensures a reasoning beyond the boundaries of one particular ruleset, they can be considered as one big knowledge source. Our experience showed that it is useful to construct separate rulesets for backward and forward chaining which allows to have each of them optimised for their specific purpose. The construction of rulesets for both forward and backward chaining is sometimes hard, because of the combination with procedural knowledge (e.g. database and CAD-interaction) called from within the body of the rules.

It must be clear that the construction of rules is not as easy as it might seem at first sight. It is certainly a misunderstanding that statements like for instance '*when there is an undercut, there must be a slide*' or '*when there is an internal undercut, one must use a lifting slide*' can be considered as rules that will lead to any result. Such rules may be perfect guidelines for a human expert but are mostly useless for a computer system. The reason is very simple: every word in a written rule like stated above has a meaning that is obvious for a human being but not for a computer program: '*what is an undercut?*' '*what defines an undercut?*' '*what are the inside and the outside of a product?*'. Formalising design knowledge encompasses more than just identifying IF-THEN expressions.

6 CONCLUSION

The various levels of information within an intelligent support system for the design of injection moulds have been described. The data of standard mould components, that are available from a number of suppliers, is stored in a relational database. The designer can access these standard catalogues and insert components into the mould. The 'real' design information is stored in an object oriented, feature based mould model. This mould model contains geometrical information of a higher level than that in the CAD database and also stores a great deal of technological information such as material codes, BOM codes, costs, suppliers, functionality. Moreover, it stores a number of different relationships between the various objects in a mould. The dynamic nature of objects allows to do complex reasoning through complex data manipulation and so the mould model contains considerable amount of procedural knowledge. The design expertise, that is of a more changing nature, is represented in IF-THEN type rules which are gathered in different knowledge sources. The system that has been described contains knowledge sources for product definition, cavity layout, cooling, demoulding mechanisms, inserts and cost estimation. A blackboard control software acts like a moderator, selecting appropriate knowledge sources. This combination of object oriented programming techniques and a rule based expert system implemented in LISP showed to be a powerful tool to support the design of injection moulds.

7 REFERENCES

Coyne, R.D., Roseman, M.A., Radford, A.D., Balachandran, M and Gero, J.S. (1990) Knowledge-based Design Systems. Addison-Wesley, Sydney.

Giarratano, J. and Riley, G. (1989) Expert Systems, Principles and Programming. MA, PWS-Kent, Boston.

Kruth, J.P. and Kesteloot, P. (1989) CAD/CAM reinforces the competitive edge of European mould makers, Proceedings of 9th MICAD Conference, Paris.

Kruth, J.P. and Willems R. (1994) Intelligent Support System for the Design of Injection Moulds, Journal of Engineering Design, Vol. 5, No. 4, pp. 339-351.

Kusiak, A. Szczerbicki, E. and Vujosevic, R (1991) Intelligent design synthesis: an object-oriented approach, Int. Journal Prod. Res., Vol. 29, No. 7, pp 1291-1308.

Lecluse, D. and Sleeckx, E. (1993) An integrated design support system for mechanical applications applied for plastic injection mould design, Proceedings of 7th International Conference on Systems Research, Informatics and Cybernetics.

Menges, G. and Mohren, P. (1986) How to Make Injection Moulds. Hanser, New York.

Meyer, B. (1988) Object-Oriented Software Construction. Prentice Hall.

Pye, R.G.W. (1989) Injection Mould Design. Longman, London.

Salomons, O.W., van Houten, F.J.A.M., Kals, H.J.J. (1992), Review of research in Feature-Based Design, Journal of Manufacturing Systems, Vol. 12, No. 2., pp 113-132.

Schwarz, H. (1985) Catalogue of standardized CAD parts - an essential component of economical CAD/CAM application in the design of tools and moulds, Industrial & Production Engineering, 3, pp. 104-113.

Shah, J.J. (1990) An Assessment of features technology, proceedings of the CAM-I feature symposium P-90-PM-02, pp 55-77.

Stroustrup, B. (1988) What is Object-Oriented Programming?, IEEE Software, Vol. 5, No. 3, pp 10-20.

Wierda, L.S. (1991), Linking Design, Process Planning and Cost Information by Feature-based Modelling, Journal of Engineering Design, Vol. 2, No. 1, pp 3-19.

Winston, P.H. (1984) Artificial Intelligence. Addison-Wesley, Sydney.

Zeid, I. (1991) CAD/CAM Theory and practice. McGraw-Hill, Singapore.

8 BIOGRAPHY

Robin Willems studied electro-mechanical engineer at the dept. of Mechanical Engineering of the Katholieke Universiteit Leuven (K.U.Leuven, 1991) and he currently works there as a research engineer at the division of Production Engineering, Machine Design and Automation (PMA). He was involved in the BRITE PROMISES project (BE-3148), where he was responsible for the implementation of the expertise about plastic injection mould design and manufacturing. Currently he is involved in the European CRAFT project IMES (CR-1309) and he is preparing his Ph.D. on *Computer Aided Design and Manufacturing of Plastic Injection Moulds.*

Dirk Lecluse studied information technology at the dept. of Computer Science of the K.U.Leuven (1990) and works now as a research engineer at the Scientific and Technical Center of the Belgian Metalworking Industry (WTCM/CRIF). During the BRITE PROMISES project (BE 3148), he was responsible for the design and implementation of an expert system for plastic injection mould design. Currently he is technical project leader of the European CRAFT project IMES (CR-1309). IMES is a joint initiative of 15 industrial companies and 4 R&D partners for the development of an Injection Mould Engineering System.

Prof.dr.ir. J.P. Kruth obtained his Ph.D. at the K.U.Leuven in 1979 with a dissertation on *Computer Adaptive Control for Electro-Discharge Machining*. He worked as expert in production engineering at the Institut Technologi Bandung, Indonesia, from 1979 to 1982. Later, he became research engineer at WTCM/CRIF, till 1987. During the same period he was half-time consultant engineer at the national 'Stand-By CAD/CAM' service for industry and he lectured CAD at the K.U.Leuven. He was nominated professor at K.U.Leuven in 1987, where he chairs the division PMA. He is active member of the Institution for Production Engineering Research (CIRP), senior member of the North American Manufacturing Research Institution (NAMRI/SME), honorary member of the Rumanian Society of Mechanical Engineers, and member of several other professional associations, editorial boards and others.