Int J Adv Manuf Technol (2001) 17:549–557
© 2001 Springer-Verlag London Limited

The International Journal of
Advanced
Manufacturing
Technology

A Prototype Knowledge-Based System for Conceptual Synthesis of the Design Process

W. Y. Zhang, S. B. Tor and G. A. Britton

Design Research Center, School of Mechanical and Production Engineering, Nanyang Technological University, Singapore

This paper presents a knowledge-based system "KBCS", the "Knowledge-based conceptual synthesiser" that supports the synthetic phase of conceptual design. It was developed using an expert system shell called CLIPS. By using this knowledge, physical behaviour can be derived from a desired function or desired behaviour, and a functional model which represents causal relationships among the functions and behaviours can be created. In addition, complicated desired functions which cannot be matched with the functional output of any behaviour after searching the object-oriented behaviour base, will be automatically decomposed into subfunctions by means of relevant function decomposition rules. A case study for the conceptual synthesis of an automatic assembly system provides an application of this intelligent design environment, and a demonstration of its methodology. In this paper we also describe how two popular AI representation techniques, object-oriented representation and production rule representation, can be usefully integrated to solve the problem.

Keywords: Conceptual design; Conceptual synthesis; Expert system; Knowledge-based; Object-oriented

1. Introduction

Conceptual design is the initial and most abstract stage of the design process, starting with a desired specification and resulting in concept variants (preliminary system configurations). Conceptual synthesis during conceptual design is different from concrete configuration design undertaken during detailed design. The latter can be thought of as a process of generating artifacts by assembling predefined components with detailed information. However, conceptual design, being the early stage of design, is characterised by information that is often imprecise, inadequate and unreliable. According to Pahl and Beitz [1], conceptual design involves a great deal of time and knowledge. More importantly, a poorly conceived design concept can never be compensated for by a good detailed design.

Owing to recent advances in the field of artificial intelligence (AI), knowledge-based systems have demonstrated their capabilities by providing successful solutions in many application areas. Knowledge-based systems represent an alternative to conventional systems, particularly in the areas of symbolic reasoning and advisory tasks. According to Green [2], computers currently play two roles in the design process. One set of tools provide aid in the final drafting of the specifications, and the second provide aid in analysis. He coins the term "knowledge-aided design" (KAD) to distinguish it from the current computer-aided design (CAD) tools. Whereas CAD tools are used only after the major design decisions have been made, KAD systems operate at a much earlier phase in the design process.

Knowledge-based expert systems for conceptual design have been an active area of research for the past two decades. In the conceptual synthesis domain, a knowledge-based system is used to solve modelling and reasoning problems. The most common forms of knowledge representation include rules, frames, objects, and cases. The rule-based paradigm has been adopted by Li et al.[3] to automate the computational synthesis of the conceptual design of mechanisms. The design algorithm employs the best-first heuristic searches in a library of mechanical devices represented and classified qualitatively. Besides rule representation, frame representation is also widely used. Tong and Gomory [4] use a frame-based structure to model parts of standard kitchen appliances. They use a goal driven strategy. Arpaia et al. [5] made use of both data driven and goal driven reasoning approaches for the automatic design of measurement systems, in mapping from the logical attributes to the physical components of the instrument. Moulianitis et al. [6] presented a knowledge-based system for the conceptual design of grippers for handling fabrics with its reasoning strategy based upon a combination of a depth-first search method and a heuristic method. The heuristic search method obtains a final solution from a given set of feasible solutions and can synthesise new solutions to accomplish the required specifications. A refinement to the idea of a goal-directed search is the distinction between constraints and objectives. A constraint is a statement about a design, the truthfulness of

Correspondence and offprint requests to: Dr S. B. Tor, School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue, 639798 Singapore. E-mail: msbtor@ntu.edu.sg

which does not depend on any trade-offs with goals. Gelsey et al. [7] presented a model constraints strategy for the commutation of information about model assumption violation between a simulator and an automated search procedure which is exploring a space of candidate designs in the domain of the conceptual design of supersonic transport aircraft. An increasingly popular modelling representation is the object-oriented representation. According to Ringland [8], it has many advantages over the traditional frame-based representation. Akagi and Fujita [9] used object-oriented architecture for supporting a functional design process. The model for the design process is constructed using networks composed of knowledge elements which are represented modularly in objects. This modelling results in determining of design variables flexibly in the conceptual design process. Mao et al. [10] used cases to support a case prototype based design. Its case prototype is the class of cases using the object-oriented approach. The structure of the case prototype is used as an index structure for a case-based scheme and for a case retrieval strategy.

The objective of this research project is to develop an expert system for conceptual synthesis. The system is based on the functional reasoning process. Physical behaviour can be inferred from a desired function or desired behaviour, and a model which represents causal relationships among the functions and behaviours can be created. The behaviour representation is then used to select and arrange embodiments (abstractions of physical artifacts) to develop system configurations, which are the end result of the conceptual design process. Interconnection of these behaviours is possible when there is compatibility between the functional output of the one and the corresponding driving input of the next one. Of course, connectivity must satisfy all the constraints imposed in the problem domain. In a knowledge-based conceptual synthesiser (KBCS), a distinct solution search strategy is adopted. The inference engine always scans the object-oriented behaviour base to search for the matching behaviour whose functional output matches the desired function as the starting point. If no matching behaviour can be found, the desired function will be automatically decomposed into subfunctions by means of a certain domain specific function decomposition rule. This search strategy can prevent the domain problem being decomposed "too fine", which will cause a combinatorial explosion.

KBCS is built entirely using an object-oriented representation scheme. All the necessary declarative and procedural knowledge is embedded in objects. A behaviour base is developed on the basis of an object-oriented representation scheme. In addition, the production rules, which are bearers of heuristic knowledge, are combined with the scheme via a unified programming environment.

A case study for the conceptual synthesis of an automatic assembly system for manufacturing electronic connectors is used to demonstrate the methodology and application of KBCS.

2. Conceptual Synthesis Process

Conceptual design [1] is that part of the design process in which, by the identification of the essential problem through abstraction, by the establishment of function structures, and by the search for appropriate solution principles and their combinations, the basic solution path is established through the elaboration of a solution concept. Figure 1 shows the steps involved. We believe that conceptual synthesis, as a process of functional modelling and reasoning, should cover most of the above steps. Our work on knowledge-based conceptual synthesis is based on the systematic methodology and the development of a distinct solution search strategy.

To describe the conceptual synthesis process we need to express function, behaviour, structure, and their relationships. The basis of of function–behaviour–structure is to carry out the transition from function to structure via the synthesis of physical behaviour relating to the required function. Behaviour characterises the implementation of a function. For example, if a function *converting an action of force into the movement of an object* is required, then possibly Hooke's law for the behaviour of a spring is exploited. Here, the behaviour of a physical structure *spring* bridges the gap between the required function and the solution structure. We can see that behaviour is both of function and structure, thus forming the bridge between them. Welch and Dixon [11] defined conceptual design as the transition between three different information stages:

1. A set of required functions.
2. A set of behaviours that fulfil the functions.
3. A set of preliminary systems that produce the behaviours.

They explain that function is what a design is going to do, whereas behaviour is how it will do it. Deng et al. [12] employed a behavioural scenario of the desired product, which not only describes function, structure, and the behavioural process of the product, but, also, explicitly describes its working environment. The approach addresses the problem from a

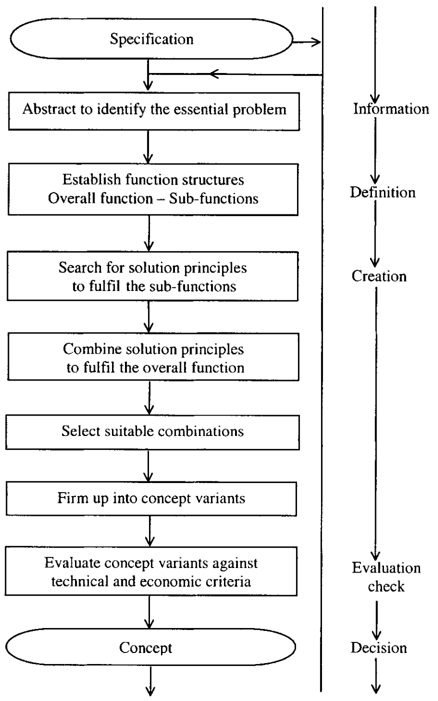

Specification
Abstract to identify the essential problem
Establish function structures
Overall function – Sub-functions
Search for solution principles
to fulfil the sub-functions
Combine solution principles
to fulfil the overall function
Select suitable combinations
Firm up into concept variants
Evaluate concept variants against
technical and economic criteria
Concept
Information
Definition
Creation
Evaluation
check
Decision

Fig. 1. Steps of conceptual design [1].

system viewpoint – not just considering the product itself, but the product system. They represent behaviour by a driving input (the intended input action), a functional output (the intended output action), and also harmful inputs and the side-effects.

Umeda et al. [13] developed a function–behaviour–state (FBS) modeller using a function prototype to represent function decomposition knowledge, and used physical features to represent knowledge of physical phenomena. Both physical state transition and physical phenomena are used to represent the behavioural process. The FBS modeller regards functional decomposition as of two types: causal decomposition; and task decomposition. With this distinction, they show that the decomposition knowledge in the function prototype includes only the task decomposition, and the function–behaviour relationship in the function prototype stores a description of the physical features, which consists of physical components and physical phenomena occurring on the components. Causal decomposition of function is used in deriving candidates for additional physical features that satisfy the conditions for the physical features identified from the function–behaviour relationship. Deng et al. [12] argued that causal decomposition of function should be more appropriately extended by the behavioural process generation task. This is because the process requires the designer to view from the behavioural scenario viewpoint, not just from the functional viewpoint. Only a behavioural scenario can provide the designer with the complete information to develop the causal structure of the required behavioural process. Hence, Deng et al. suggest a dual-step modelling procedure – initial function decomposition and causal behavioural process generation, where the causal behavioural process generation can be of several levels.

The conceptual synthesis methodology in this paper is based on the above function decomposition and causal behavioural process, but differs in the following ways:

1. For the desired function, KBCS always searches for behaviour whose functional output matches the desired function as a starting point and that also satisfies the operational constraints. When no direct match exists, the representation scheme allows for automatic decomposition of the desired function into subfunctions to facilitate further behavioural synthesis. Here, behaviour is represented in terms of driving input, behaviour actor (structure), functional output, and side effect.
2. We build a hybrid knowledge base integrating object-oriented representation and production rule representation to realise functional modelling and behavioural reasoning.

The architecture of the knowledge-based expert system including its representation and inference will be discussed in Section 3.

Let us illustrate the significance of searching for the matching behaviour whose functional output matches the desired function as a starting point, instead of decomposing the desired function into subfunctions as a starting point. For example, if the desired function is not very complicated and there is already matching behaviour which can solve the domain problem efficiently, decomposing the desired function as a starting point will cause a combinatorial explosion and prevent evaluation within a reasonable amount of time. Though the synthesis process is guided by a knowledge-based expert system, most of the combinations may prove totally useless in terms of the desired function.

With the above search strategy, a knowledge-based conceptual synthesis reasoning strategy will be introduced in next section.

3. Architecture of KBCS

3.1 Overview of KBCS

KBCS was developed using an expert system shell, CLIPS (C Language Integrated Production System). It was developed by the Software Technology Branch, NASA/Lyndon B. Johnson Space Center. CLIPS was designed to facilitate the development of software to model human knowledge or expertise. In addition to the one mode of knowledge representation, i.e. rules, knowledge representation is also available through objects using COOL (CLIPS Object Oriented Language) and Deffunctions (CLIPS Functions).

Figure 2 shows the main modules that make up KBCS. It is composed of the following modules: user interface; working memory; knowledge-base editor; knowledge base; and inference engine.

1. The user interacts with KBCS through a user interface, which employs question-and-answer, menu-driven, and GUI styles.
2. Working memory refers to the case-specific data: the facts, conclusions, and other considerations. This includes the data given in the problem instance, partial conclusions, and dead ends in the search process. This information is separate from the general knowledge base.
3. The knowledge-base editor can assist in the addition of new knowledge, help to maintain correct rule syntax, and perform consistency checks on the updated knowledge base.
4. The heart of KBCS is a hybrid knowledge base which integrates a rule base and an object-oriented behaviour base effectively. Rules can be used to express causal knowledge involving several objects, and an object can also encapsulate rules.
5. The inference engine applies the knowledge in the knowledge base to the case-specific data to arrive at some solution or conclusion. It is the interpreter for the knowledge base.

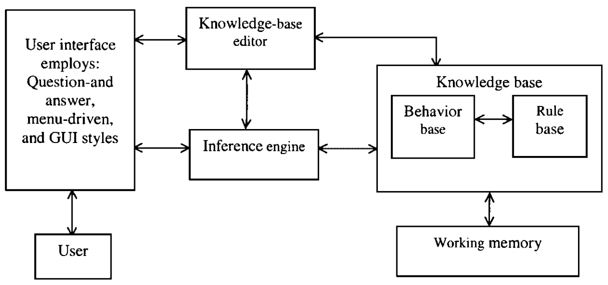


Fig. 2. Architecture of KBCS.

3.2 Knowledge Base Representation

3.2.1 Object-Oriented Knowledge Representation and Object-Oriented Behaviour Base

An increasingly popular knowledge representation is the object-oriented representation. Object-orientation is usually both a language feature and a design methodology [14]. An object is an entity that combines its data structure and its methods into one object. The characteristics of object-oriented representation are: abstraction (focus on what it does before deciding how to implement it); encapsulation (information hiding, that is, the object hides its internal structure from its surroundings); polymorphism (the sender of a stimulus does not need to know the receiving instance's class); and inheritance (of both data structure and methods which allow sharing without redundancy). Employing the object-oriented approach affords several benefits compared to traditional methods [15–17]. Kaindl [18] claimed that object-oriented modelling should be a foundation for knowledge system development, as it is a foundation for software engineering development.

KBCS is built entirely using an object-oriented representation scheme. Object-oriented techniques provide the modelling flexibility needed for conceptual design. The parameters and properties of behaviours in the behaviour base can be represented as objects and their slots. This ability to mix and combine different objects allows us to generate many design alternatives quickly.

A behaviour base is developed as an object-oriented knowledge base. The most generic behaviours can be represented as the top-most generic class object. This class of object is defined as follows:

```
Class Behaviour {
  slot:
    Name:
    Structure:
    Driving_Input:
    Functional_Output:
    . . . . . .
  method:
    Input_Data ( );
    Output_Data ( );
    Anti_Loop ( );
    Search_Branch_End ( );
    . . . . . .
}
```

Here, the generic class *Behaviour* encapsulates some slots and methods. The slot *Name*, *Structure*, *Driving_Input*, and *Functional_Output* shown in the above pseudo-code are, respectively, the name, structure, driving input, and functional output of defined behaviour. Note that the slot *Structure* does not mean that only the structure name can be incorporated. In fact, it is the schematic geometric structure. Figure 3 shows the schematic geometric structure of some behaviours in a case-specific domain. Method *Input_Data ( )* will facilitate the user to input the required concrete values to the slots. Method *Output_Data( )* will return the relevant slot values to the working memory of the KBCS. Method *Anti_Loop ( )* will

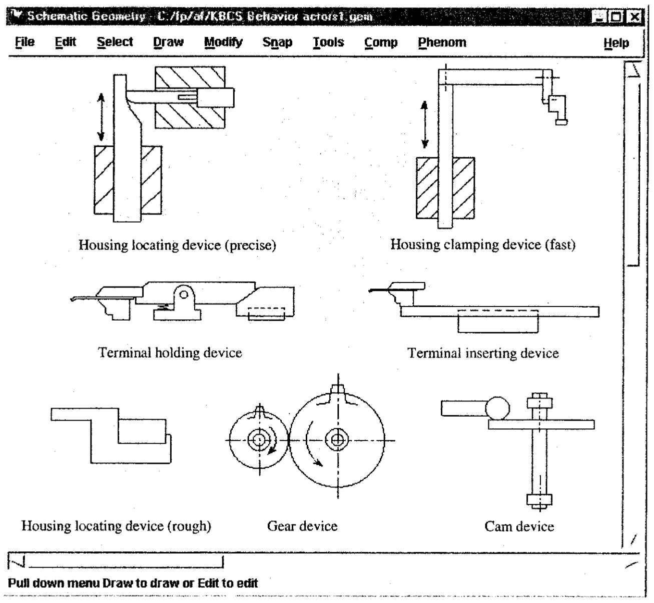

Schematic Geometry - C:/lp/al/KBCS Behavior actors1.gem
File Edit Select Draw Modify Snap Tools Comp Phenom Help
Housing locating device (precise)
Housing clamping device (fast)
Terminal holding device
Terminal inserting device
Housing locating device (rough)
Gear device
Cam device
Pull down menu Draw to draw or Edit to edit

Fig. 3. Schematic geometry of some behaviour actors (structures).

invoke the rule *Anti_Loop* when this behaviour object is found. Rule *Anti_Loop* is an anti-looping rule and will be introduced in Section 3.2.2. Method *Search_Branch_End ( )* will invoke the rule *Search_Branch_End* when this behaviour object is retrieved to working memory. Rule *Search_Branch_End* is used to end the search branch and will be introduced in Section 3.2.2.

The other kind of behaviour class which includes the *Side_Effect* slot can be represented as the child class object of the generic class *Behaviour*. It can inherit the latter's slots and methods, and can also add specific slots and methods pertinent to the child class object. For example, this child class can be represented as:

```
Class Behaviour_With_Side_Effect {
  Inherit: Behaviour
slot:
  Side_Effect:
  . . . . . .
method:
  Prevent_Side_Effect ( );
  . . . . . .
}
```

Here, the slot *Side_Effect* defines the side effect of the defined behaviour. The method *Prevent_Side_Effect ( )* can prevent this side effect by searching for and retrieving other behaviour objects whose functional output can prevent the former behaviour's side effect.

Both the above behaviour objects are represented by a class. A class is only a template for the structure of objects (including slots and methods). They only encapsulate slots and methods, not concrete data. We can represent the concrete behaviours by applying instances to these classes. An example instance which uses the above class *Behaviour_With_Side_Effect* as a represented class is listed below:

```
Instance Terminal_Insert_Device {
  Class Behaviour_With_Side_Effect
  slot:
    Name = “Terminal inserting device”
    Structure = Schematic geometry
    Driving_Input = “Provide translational motion”
    Funtional_Output = “Insert terminal”
    Side_Effect = “Terminal moves too much”
    . . . . . .
  method:
    Input_Data( );
    Output_Data( );
    Anti_Loop( );
    Search_Branch_End( );
    Prevent_Side_Effect( );
    . . . . . .
```

The first and second lines above are used to define an instance of class *Behaviour_With_Side_Effect*. The lines that follow are used to assign data to slots or activate methods for this instance. Thus, the class object is fully represented by an instance object.

The relationships between two above-mentioned behaviour class objects and an instance object are illustrated in Fig. 4.

In Fig. 4, a class at a lower level inherits some slots of the class at the higher level. Each instance can access these slots from the higher level by inheritance. When new behaviour is added to the behaviour base, it is compared with the objects that are already in the behaviour base. If it is the same as a class that already exists, it can be represented as an instance of that class. If it is similar to a class, but has different parts, it is necessary to build a new object. In this case, the different parts are described in the new class, whereas the similar parts are derived from the class of the higher level by inheritance. Thus, object-oriented representation is very convenient for the maintenance and modification of the behaviour base.

3.2.2 Rule-Based Representation

Though an object-oriented system provides convenient ways to represent complex data structures, the rule-based system in KBCS allows intuitive expression of relationships among data items (through rules) and facilitates symbolic processing of those rules to determine what conclusions may be drawn from the data. Rules in KBCS include domain specific and general production rules which are presented next.

3.2.2.1 Domain Specific Rules. Domain specific rules refer to a set of rules that are used to solve domain dependent problems. In KBCS, we will use the proposed methodology for designing an automatic assembly system for manufacturing electronic connectors. In this application domain, examples of the domain specific rules are formulated as follows:

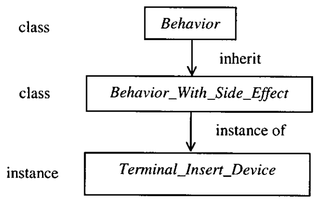


Fig. 4. Hierarchy of behaviour objects.

Rule *Specific_Decompose1*
  IF a desired function is *Insert terminal into housing*
  THEN decompose it into *Clamp housing after locating it*
  AND *Insert terminal after holding it*

Rule *Specific_Decompose2*
  IF a desired function is *Clamp housing after locating it*
  THEN decompose it into *Locate housing*
  AND *Clamp housing*

Rule *Specific_Decompose3*
  IF a desired function is *Insert terminal after holding it*
  THEN decompose it into *Hold terminal*
  AND *Insert terminal*

In the case that the desired domain specific function is too complicated to find a matching behaviour directly, after searching the behaviour base, the desired function should be broken down into less complex subfunctions by means of some domain specific function decomposition rules such as the above-mentioned rules *Specific_Decompose1*, *Specific_Decompose2*, and *Specific_Decompose3* to facilitate the subsequent search for causal matching behaviour.

3.2.2.2 General Rules. General rules refer to a set of rules that are used to solve general problems. KBCS can be applied to various application domains after changing the domain-specific behaviour base and the domain specific rules. Some general production rules are formulated as follows:

Rule *Not_Decompose*
  IF a function is matched with a behaviour in behaviour base
  OR is matched with an environment element
  THEN do not decompose it

Rule *Decompose*
  IF a function is not matched with a behaviour in behaviour base
  OR is not matched with an environment element
  THEN decompose it

The above rules *Not_Decompose* and *Decompose* are used to decide when to decompose a function.

Rule *Anti_Loop*
  IF last behaviour object found belongs to a previous selected list
  THEN terminate search

This is an anti-looping rule without which some behaviour objects may be called recursively, so this rule is applied every time a behaviour object is found.

Rule *Search_Branch_End*
  IF all driving inputs of behaviour objects are available in environment
  AND side effects of behaviour objects are successfully prevented
  THEN terminate branch
  AND start searching next branch

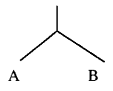
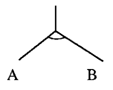
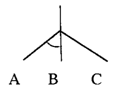
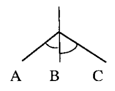
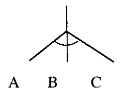
AND/OR clause representation	
Graphical	Literal
	A OR B
	A AND B
	A AND B OR C
	A AND B OR B AND C
	A AND B AND C

Fig. 5. Graphical and literal representation of AND/OR clauses [19].

This rule is used to terminate a branch and results in feasible concept variants. This rule is applied every time a behaviour object is retrieved to the working memory.

3.3 Inference Engine and Search Strategy

The distinct search strategy of KBCS includes data driven and goal driven approaches. The procedures that implement the control cycle are separated from the knowledge base. Before the analysis of the distinct search strategy of KBCS, we adopt the notation specified by Kusiak and Szczerbicki [19], which is shown in Fig. 5.

According to this notation, Fig. 6 shows an illustrative example of search strategy of KBCS. Recall from Section 3.2.2 that there are some domain specific rules and general rules for knowledge representation, and these production rules will be quoted here. The starting point of inferencing strategy is to put the goal function *F1* in the working memory and scan the behaviour base to seek the behaviour whose functional output can match function *F1*. It is supposed that no matching is found after scanning the whole behaviour base. Then the inference engine scans the rule base to search for the problem-solving production rules. Because the premise *a function is not matched with a behaviour in behaviour base* is satisfied, the general rule *Decompose* is fired to make the *Decompose it* decision. Now it is supposed the premise of one domain specific function decomposition rule is satisfied by goal function *F1*, then this rule will be fired and its conclusion puts functions *F11* and *F12* in the working memory as subgoal functions. Here, the search strategy for function decomposition employs the data-driven control regime.

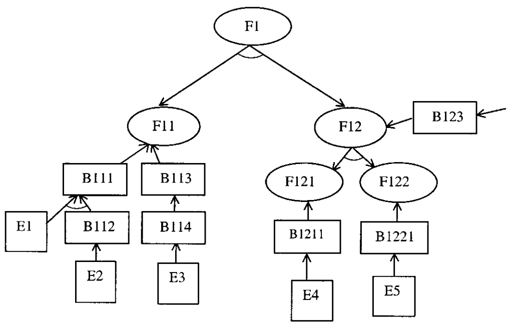


Fig. 6. Search tree of KBCS.

For function *F11*, the starting point of the inferencing strategy is to scan the behaviour base to search for one behaviour whose functional output can match this desired subfunction *F11*. It is supposed that functional output of either behaviours *B111* and *B113* can match *F11*. Then, behaviours *B111* and *B113* are retrieved to the working memory, and their driving inputs are taken to be the new design goals. Suppose behaviour *B111* has two driving inputs: *Driving_Input1* and *Driving_Input2*. Because *Driving_Input1* is available in the environment *E1*, the general rule *Search_Branch_End* is fired to terminate this branch. Suppose the functional output of behaviour *B112* can match the *Driving_Input2* of behaviour *B111* when the inference engine scans the behaviour base, behaviour *B112* is retrieved to the working memory. Because the driving input of behaviour *B112* is available in environment *E2*, the general rule *Search_Branch_End* is fired to terminate this branch. This search strategy for the causal behavioural process employs a goal-driven control regime. The strategy retrieves *B111* and *B112* to achieve *F11*. Similarly, alternative behaviours *B113* and *B114* are found to achieve subfunction *F11*.

For function *F12*, the starting point of the inferencing strategy is to scan the behaviour base to seek the behaviour whose functional output can match this desired subfunction *F12*. It is supposed that the behaviour *B123* is retrieved to achieve subfunctions *F12* and its driving input is now taken to be the new design goal. The inference engine continues to scan the behaviour base to seek the behaviour whose functional output can match the driving input of behaviour *B123*. It is supposed that no matching is found after scanning the whole behaviour base. Then, the inference engine terminates and discards this searching branch, and returns to function *F12* to scan the problem solving production rule base to search for a matching rule. Suppose one domain specific function decomposition rule is fired, and its premise is *F12* and its conclusions are *subfunctions F121 and F122*. The process continues, and behaviour *B1211* and *B1221*, which, respectively, achieve subfunction *F121* and *F122*, are developed.

During the above exhaustive search strategy, after every search step, the inference engine will check constraints and the environment to decide whether to terminate the processing search branch and check if there are any unexplored branches. If all search branches have been explored, the run is terminated. The concept variants produced by the run will be listed. Behaviours in a pair of parentheses can achieve a certain goal function or subfunction with their end driving inputs satisfied by the environment. The resulting variants for the example above are shown below:

Variant 1 → (B111 + B112) + (B1211) + (B1221)
Variant 2 → (B113 + B114) + (B1211) + (B1221)

4. Case Study

KBCS has been used successfully for designing the automatic assembly system for manufacturing electronic connectors. The automatic assembly system for manufacturing connectors comprises of a vibrator bowl feeding unit, a housing singulator, a walking beam unit, a terminal inserting unit, a terminal cutting unit, a terminal bending unit, a carrier removing unit, an HY-pot test unit, a contact continuity test unit, a reject station, and an automatic offloading unit. Among them, the terminal inserting unit is the main and most complicated unit, so this case study will be focused on the conceptual synthesis for the terminal inserting unit of this automatic assembly system.

4.1 Problem Description

Suppose the following specification is given:

1. Design the terminal inserting unit whose overall function is *Insert terminal into housing*.
2. The environment can provide the following functional outputs:

   E1's functional output: *Provide pneumatic air*.
   E2's functional output: *Provide electric power*.
   E3's functional output: *Fix the device.*
3. The following constraint applies:
   The inserting position tolerance $<$ *0.1 mm*. (which means *High precision location is needed*).

4.2 Problem Solving Strategy

Recall from Section 3.2.2 that there are some domain specific and general rules for knowledge representation. These production rules will be quoted in this case study. Referring to Fig. 7, the logical steps of the inference engine are the following:

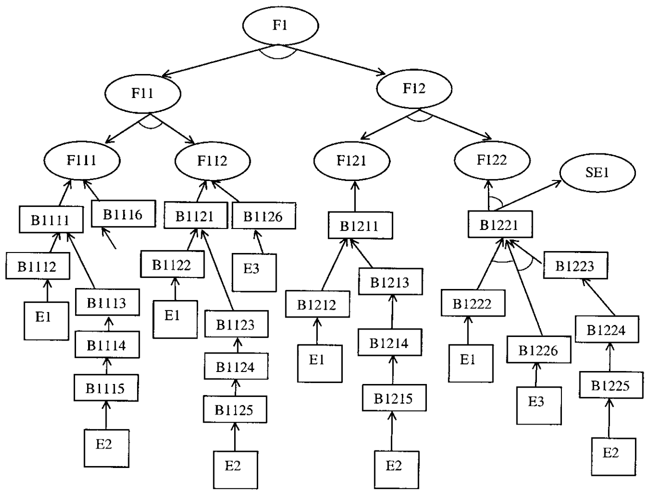


Fig. 7. Search tree of KBCS in case study.

1. The starting point of inferencing strategy is to put the goal function *F1* in the working memory and scan the behaviour base to seek the behaviour whose functional output can match function *F1*. Because no matching is found after scanning the whole behaviour base, the inference engine starts to scan the rule base to search for the problem-solving rules. Because the premise *a function is not matched with a behaviour in behaviour base* is satisfied, the general rule *Decompose* is fired to make the *Decompose it* decision. Then, with its premise desired function *F1* satisfied, domain specific rule *Specific_Decompose1* is fired to decompose function *F1* into subfunctions *F11* and *F12*.
   Where:

   F1: *Insert terminal into housing.*
   F11: *Clamp housing after locating it.*
   F12: *Insert terminal after holding it.*
2. Similarly, *F11* is decomposed into *F111* and *F112* with rule *Specific_Decompose2* activated.
   Where:

   F111: *Locate housing.*
   F112: *Clamp housing.*
3. For *F111*, the starting point of the inferencing strategy is to scan the behaviour base to search for one behaviour whose functional output can match the desired subfunction *F111*. It is found that either functional outputs of behaviours *B1111* and *B1116* can match *F111*. Then, the behaviour *B1111* and *B1116* are retrieved to the working memory, and their driving inputs are, respectively, taken to be the new design goals. Similarly, the behaviour *B1112* is developed with its functional output *Provide translational motion* matching the behaviour *B1111*'s driving input *Provide translational motion*. Now, the behaviour *B1112*'s driving input *Provide pneumatic air* becomes the new goal. Because the environment *E1* can satisfy *Provide pneumatic air*, the general rule *Search_Branch_End* is fired, and this searching branch is terminated and put in the configuration list.
   Where:

   B1111: *Housing locating device (precise).*
   B1112: *Cylinder device.*
   B1116: *Housing locating device (rough).*
4. Similarly, *B1111*'s driving input can be matched by *B1113*'s functional output *Provide translational motion*; *B1113*'s driving input *Provide low speed rotary motion* can be satisfied by *B1114*'s functional output *Provide low speed rotary motion*; *B1114*'s driving input *Provide high speed rotary motion* can be satisfied by *B1115*'s functional output *Provide high speed rotary motion*; and *B1115*'s driving input *Provide electric power* can be satisfied by environment *E2*. This search branch is terminated and put in the configuration list.
   Where:

   B1113: *Cam device.*
   B1114: *Gear pair device.*
   B1115: *Motor device.*

5. Though behaviour *B1116*'s functional output can match *F111*, it is rejected because it does not meet the constraint requirement: *High precision location is needed*.

   Now the causal behavioural searching process for realising function *F111*: *Locate housing* is finished with two feasible branches being developed. Figure 8 shows a detailed process representation.

6. The causal behavioural branches for *F112* can be developed by means of scanning the behaviour base and retrieving the matching behaviours.
   Where:

   B1121: *Housing clamping device (fast).*
   B1121's driving input: *Provide translational motion.*
   B1121's functional output: *Clamp housing fast.*
   B1122: *Cylinder device.*
   B1122's driving input: *Provide pneumatic air.*
   B1122's functional output: *Provide translational motion.*
   B1123: *Cam device.*
   B1123's driving input: *Provide low speed rotary motion.*
   B1123's functional output: *Provide translational motion.*
   B1124: *Gear pair device.*
   B1124's driving input: *Provide high speed rotary motion.*
   B1124's functional output: *Provide low speed rotary motion.*
   B1125: *Motor device.*
   B1125's driving input: *Provide electric power.*
   B1125's functional output: *Provide high speed rotary motion.*
   B1126: *Housing clamping device (slow).*
   B1126's driving input: *Fix the device.*
   B1126's functional output: *Clamp housing slowly.*

7. *F12* can be decomposed into *F121* and *F122* with rule *Specific_Decompose3* activated.

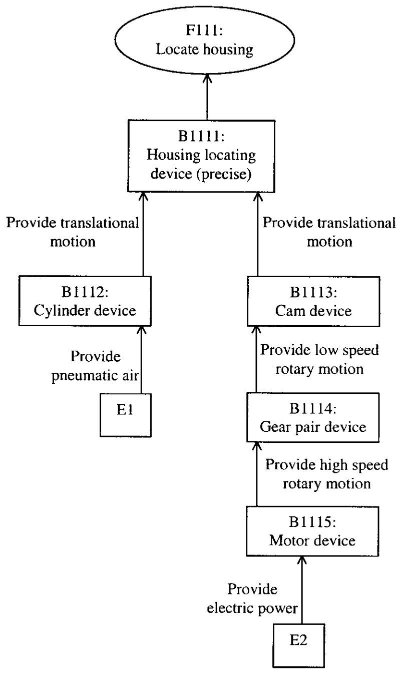

F111:
Locate housing
B1111:
Housing locating device (precise)
Provide translational motion
Provide translational motion
B1112:
Cylinder device
B1113:
Cam device
Provide pneumatic air
Provide low speed rotary motion
E1
B1114:
Gear pair device
Provide high speed rotary motion
B1115:
Motor device
Provide electric power
E2

Fig. 8. Partial detailed causal behavioural searching process.

   Where:

   F121: *Hold terminal.*
   F122: *Insert terminal.*

8. The causal behavioural branches for *F121* and *F122* can be developed by means of scanning the behaviour base and retrieving the matching behaviours.
   Where:

   B1211: *Terminal holding device.*
   B1212: *Cylinder device.*
   B1213: *Cam device.*
   B1214: *Gear pair device.*
   B1215: *Motor device.*
   B1221: *Terminal inserting device.*
   B1222: *Cylinder device.*
   B1223: *Cam device.*
   B1224: *Gear pair device.*
   B1225: *Motor device.*
   B1226: *Stopper.*

   The explanation for the side effect *SE1*: *Terminal moves too much* is noted below:

   The behaviour *B1221* is developed to match the function *F122*, but the behaviour *B1221* simultaneously produces the side effect *SE1* which should be prevented. So KBCS automatically scans the behaviour base to search for the behaviour whose functional output can prevent the behaviour *B1221*'s side effect *SE1*. Then, behaviour *B1226* is retrieved with its functional output being *Prevent terminal moving too much*, and the behaviour *B1226*'s driving input is available in environment *E3*.

9. Check if there are any unexplored branches. If there are none, terminate the run. A list of 24 theoretically feasible concept variants produced by the above run are shown below:

CONCEPT VARIANTS ARE

Variant 1 → (Housing locating device (precise) + Cylinder device) + (Housing clamping device (slow)) + (Terminal holding device + Cylinder device) + (Terminal inserting device + Cylinder device + Stopper)
Variant 2 → (Housing locating device (precise) + Cylinder device) + (Housing clamping device (slow)) + (Terminal holding device + Cylinder device) + (Terminal inserting device + Cam device + Gear pair device + Motor device + Stopper)
Variant 3 → (Housing locating device (precise) + Cylinder device) + (Housing clamping device (fast) + Cylinder device) + (Terminal holding device + Cylinder device) + (Terminal inserting device + Cylinder device + Stopper)
Variant 4 → (Housing locating device (precise) + Cylinder device) + (Housing clamping device (fast) + Cylinder device) + (Terminal holding device + Cylinder device) + (Terminal inserting device + Cam device + Gear pair device + Motor device + Stopper)

Total of 24 variants generated

10. According to Pahl and Beitz [1], we will evaluate all the resulting concept variants to narrow the choice. This final

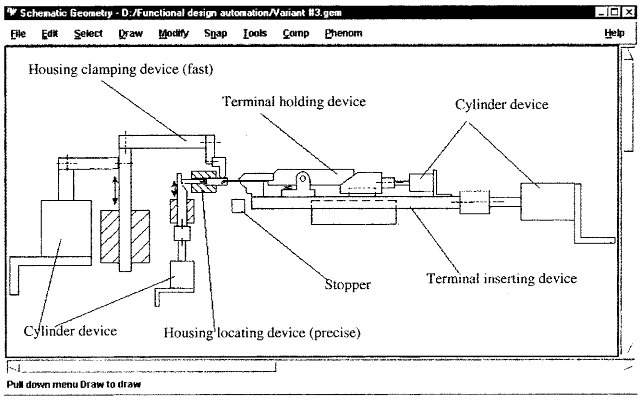

Schematic Geometry - D:/Functional design automation/Variant #3.gem
File Edit Select Draw Modify Snap Tools Comp Phenom Help
Housing clamping device (fast)
Terminal holding device
Cylinder device
Stopper
Terminal inserting device
Cylinder device
Housing locating device (precise)
Pull down menu Draw to draw

Fig. 9. Graphical representation of concept variant 3 for terminal inserting unit.

decision-making phase is the phase of concept evaluation and selection where all the concept variants generated are evaluated with respect to each other, and the highest scoring variants are selected in order of value. The values by which the concept variants are evaluated and decided upon are generated here by conducting a concept evaluation based on technical and economic criteria which are selected based on the requirements of the automatic assembly system. Variants 3 and 4 are eventually chosen as the two best concept variants. Figures 9 and 10 shows a graphical representation of Variants 3 and 4.

5. Conclusion

This paper describes a knowledge-based approach to conceptual synthesis in the conceptual design phase. The hybrid knowledge base developed includes an object-oriented behaviour base and a production rule base, which can deal with the function decomposition and the behaviour reasoning problem, cooperatively. An inference engine employs both goal driven and data driven approaches which are used in searching for causal behaviours and decomposing complex functions respectively. For the desired function, an inference engine always searches for matching causal behaviours as a starting point. This distinct searching strategy can prevent a domain problem being decomposed "too fine", which will cause a combinatorial explosion.

This knowledge-based conceptual synthesiser "KBCS" was developed using CLIPS, a declarative programming language. This prototype system also illustrates the potential of developing similar knowledge-based expert systems for practical applications.

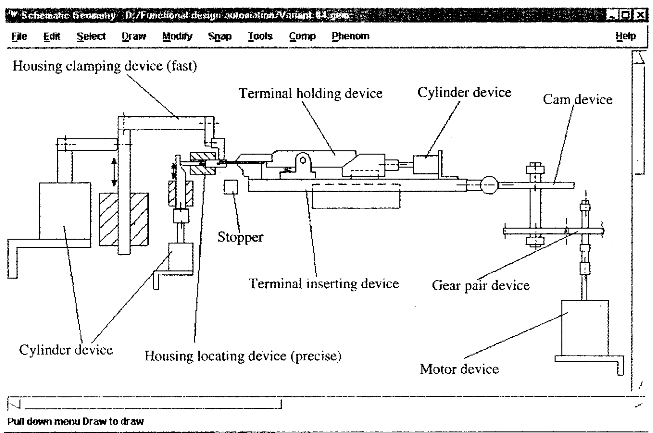

Schematic Geometry - D:/Functional design automation/Variant #4.gem
File Edit Select Draw Modify Snap Tools Comp Phenom Help
Housing clamping device (fast)
Terminal holding device
Cylinder device
Cam device
Stopper
Terminal inserting device
Gear pair device
Cylinder device
Housing locating device (precise)
Motor device
Pull down menu Draw to draw

Fig. 10. Graphical representation of concept variant 4 for terminal inserting unit.

References

1. G. Pahl and W. Beitz, in K. Wallace (ed.), Engineering Design – A Systematic Approach, London, Springer-Verlag, 1988.
2. M. Green, "Conceptions and misconceptions of knowledge aided design", Knowledge-Based Systems, 10, pp. 1–24, 1992.
3. C. L. Li, S. T. Tan and K. W. Chan, "A qualitative and heuristic approach to the conceptual design of mechanisms", Engineering Application of Artificial Intelligence, 9(1), pp. 17–31, 1996.
4. C. Tong and A. Gomory, "A knowledge based computer environment for the conceptual design of small electromechanical appliances", Computers, 26(1), pp. 69–71, 1993.
5. P. Arpaia, G. Betta, A. Langella and M. Vanacore, "Expert system for the optimum design of measurement systems", IEEE Proceedings on Science, Measurement and Technology, 142, pp. 330–336, 1995.
6. V. C. Moulianitis, A. J. Dentsoras and N. A. Aspragathos, "A knowledge-based system for the conceptual design of grippers for handling fabrics", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 13, pp. 13–25, 1999.
7. A. Gelsey, M. Schwabacher and D. Smith, "Using modeling knowledge to guide design space search", Artificial Intelligence, 101, pp. 35–62, 1998.
8. G. A. Ringland, in D. A. Duce (ed.), Approaches to Knowledge Representation: An Introduction, John Wiley, New York, 1988.
9. S. Akagi and K. Fujita, "Building an expert system for engineering design based on the object-oriented knowledge representation concept", Journal of Mechanical Design, 112, pp. 215–222, 1990.
10. Q. Mao, J. Qin, X. Zhang and J. Zhou, "Case prototype based design: philosophy and implementation", Computers in Engineering, 1, pp. 369–374, 1994.
11. R. V. Welch and J. R. Dixon, "Representing function, behaviour and structure during conceptual design", Design Theory and Methodology, 14, pp. 11–18, 1992.
12. Y.–M. Deng, S. B. Tor and G. A. Britton, "A computerized design environment for functional modeling of mechanical products", 5th ACM Symposium on Solid Modeling, Ann Arbor, Michigan, USA, pp. 1–12, 1999.
13. Y. Umeda, M. Ishii, M. Yoshioka, Y. Shimomura and T. Tomiyama, "Supporting conceptual design based on the function-behaviour-state modeler", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 10, pp. 275–288, 1996.
14. B. Meyer, Object-Oriented Software Construction, Prentice-Hall, 1988.
15. J. Rumbaugh, M. Blaha, W. Premeriani, F. Eddy and W. Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, 1991.
16. G. Booch, Object-Oriented Analysis and Design with Applications, 2nd edn, Benjamin Cummings, 1994.
17. B. Meyer, Object Success: A Manager's Guide to Object Orientation, its Impact on the Corporation, and its Use for Reengineering the Software Process, Prentice-Hall, 1995.
18. H. Kaindl, "Object-oriented approaches in software engineering and artificial intelligence", Journal of Object-Oriented Programming, 6(8), pp. 38–45, 1994.
19. A. Kusiak and E. Szczerbicki, "A formal approach to design specifications", in B. Ravani (ed.), Advances in Design Automation, ASME, pp. 311–316, 1990.