Engineering with Computers (2001) 17: 339–353
© 2001 Springer-Verlag London Limited

Engineering with
Computers

EFDEX: A Knowledge-Based Expert System for Functional Design of Engineering Systems

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

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

Abstract. *This paper presents a knowledge-based system, 'EFDEX', the Engineering Functional Design Expert, which was developed using an expert system shell, CLIPS 6.1, to perform intelligent functional design of engineering systems. On the basis of a flexible, causal and hierarchical functional modeling framework, we propose a knowledge-based functional reasoning methodology. By using this intelligent functional reasoning strategy, physical behavior can be reasoned out from a desired function or desired behavior, and interconnection of these behaviors is possible when there is compatibility between the functional output of one and the corresponding functional requirement (e.g. driving input) of the next one. In addition, a complicated, desired function which cannot be matched with the functional output of any behavior after searching the object-oriented behavior base, will be automatically decomposed into less complex subfunctions by means of relevant function decomposition rules. An intelligent system for the functional design of an automatic assembly system provides an application of this intelligent design environment, and a demonstration of its methodology. In this paper, a knowledge-based functional representation scheme which integrates two popular AI representation techniques (object-oriented representation and rule-based representation) is also proposed as a prelude to a knowledge-based functional design system.*

Keywords. Expert system; Functional design; Functional modeling; Functional reasoning; Knowledge-based; Object-oriented

1. Introduction

For years, Computer-Aided-Design (CAD) has provided the industry with advanced geometric modeling and capturing techniques, relieving designers and engineers from the mundane tasks of modeling and drafting, while improving accuracy, consistency and productivity. CAD has proved to be an invaluable tool to designers, and is well suited for the downstream stage of design. However, what is more critical is the upstream stage of design, i.e. the initial and most abstract stage of the design process, starting with a desired specification and resulting in concept variants. This is the area where CAD technology is still not well developed. Conceptual design, being the early stage of design, is characterised by information that is often imprecise, inadequate and unreliable. More importantly, a poorly conceived design concept can never be compensated for by a good, detailed design. Essentially, this stage is function-driven and function-oriented, because the main design focus at this stage is to find a design solution that is able to achieve the required functions. Functional design is a new perspective on research into this design process.

The major drawback of traditional CAD technology is that it cannot perform functional design efficiently, because it doesn't have the built-in intelligence to perform functional reasoning, and lacks the knowledge to draw conclusions from inadequate and approximate information that is available. With recent advances in the field of Artificial Intelligence (AI), particularly symbolic representation and related problem-solving methods, intelligent functional design techniques have now become possible.

The main objective of our research project EFDEX is to develop an intelligent functional reasoning methodology based on an appropriate functional modeling framework, so that computers can play a more active role in the functional design process.

In this project, we develop a flexible, causal and hierarchical functional modeling framework, on the basis of which we propose a knowledge-based functional reasoning strategy to reason out the physical behavior from a desired function or behavior. Inter-

Correspondence and offprint requests to: S. B. Tor, School of Mechanical and Production Engineering, N3 Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: msbtor@ntu.edu.sg

connection of these behaviors is possible when there is compatibility between the functional output of one and the corresponding functional requirement of the next one. Of course, connectivity must stringently satisfy all the functional constraints imposed on the problem domain. The behavior representation is then used to select and arrange embodiments (abstractions of physical artifacts) to develop a set of potential concept variants.

In this paper, a knowledge-based functional representation scheme which integrates two popular AI representation techniques, object-oriented representation and rule-based representation, is also proposed as a prelude to a knowledge-based functional design system.

In EFDEX a distinct solution search strategy is adopted. The inference engine always scans the object-oriented behavior base to search for the matching behavior whose functional output matches the desired function as a starting point. Only if no matching behavior can be found will the desired function be automatically decomposed into less complex sub-functions, by means of a certain domain-specific function decomposition rule. This search strategy can prevent the domain problem from being decomposed 'too fine', which may cause combinatorial explosion.

EFDEX was developed using an expert system shell CLIPS (C Language Integrated Production System) [1], which was developed by the Software Technology Branch, NASA/Lyndon B. Johnson Space Center. A case study for intelligent functional design of the automatic assembly system for manufacturing electronic connectors is used to demonstrate the methodology and application of EFDEX.

2. Function and Functional Design

2.1. Definition of Function

There is no uniform definition of function, with different researchers [2–4] attributing different meaning either to indicate the purpose or the action of a design. However, almost all of them indicate that there is a tight coupling between function and behavior. In general, function is *what* a design is going to do, while behavior is *how* a design will do it. Thus, we present the following understanding of function:

The function of a design system is its purpose and intention in some context, and is often expressed as functional requirements and restrained by functional constraints. Function characterizes the abstracts of behavior.

Here we also impose the functional constraint in the functional definition, which is a kind of function-related design constraint, and should be stringently satisfied in the functional design process.

2.2. Specification of Function

Functions can be formulated as pairs of *a transitive verb* and *a noun*, and sometimes *a complement* is also necessary. The task to be carried out is determined by the combined verb-noun pair and its complement. The *verb* is used to describe the action relating to the function, e.g. 'insert', 'change', etc. The *noun* is used to describe the target of the action. The *complement* is used to provide additional information to the verb-noun pair, e.g. 'greatly', 'to a great extent', etc.

A functional vocabulary thesaurus technique [5] is employed to build a functional vocabulary library consisting of commonly used or domain-specific *verbs*, *nouns* and *complements*. A hierarchical and flexible FMF classification scheme [6] provides a basis for the organization (structuring) and manipulation (creating, indexing and retrieving) of functional terms. That is, each function category corresponds with some commonly used functional vocabularies, either general- or domain-specific. These functional vocabularies are compiled and stored in the functional vocabulary library, which will be used whenever the designer needs to specify function throughout the design process.

Each instantiable function may be accessed via a number of associated keywords – the common usage terms by which they are generally known. The keywords in turn will appear in the functional vocabulary library containing a synonym table of those keywords. Using these facilities, it is possible to help the designer, by means of prompts and iteration, to establish the context of the design, and the desired functions.

2.3. Functional Design

The design process which achieves functional requirements and stringently satisfies functional constraints are referred to by us as 'functional design'. The objective of functional design is to provide computer-aided tools to link design functions with the structural (physical) embodiments used to realize the function [7].

Realizing that environment is also an important design characteristic during functional modeling, some researchers [8,9] argue that a product can only function in a certain (intended) working environment. Thus, in our proposed functional design process, we also capture aspects of the working environment so as to develop a complete functional modeling framework.

With the above understanding of function and its related design characteristics (behavior, structure, environment and constraint), we can summarize that during functional design, we may not consider any of them independently because all of them relate tightly with each other in a unified design system. Functional design (Fig. 1) starts from functional requirements and functional constraints, via behaviors which fulfil the functions and interact with the working environment, and results in mapping potential physical structures. Here, function corresponds to the abstract stated behavior, and structure corresponds to the concrete stated behavior. The potential physical structures are then evaluated and ranked to generate a best final solution concept.

3. Extended FEBS Functional Modeling Framework

As a design task can be defined as transforming the function of an artifact into its physical product description, functional modeling exists ubiquitously to model a design and requirements from its functional aspects so as to allow reasoning about its function for various activities. During functional modeling, there should be a relationship framework between function and other design characteristics (e.g. structure, behavior and environment). Using an appropriate functional modeling framework, the system can further perform a functional reasoning process on the basis of predetermined relationships.

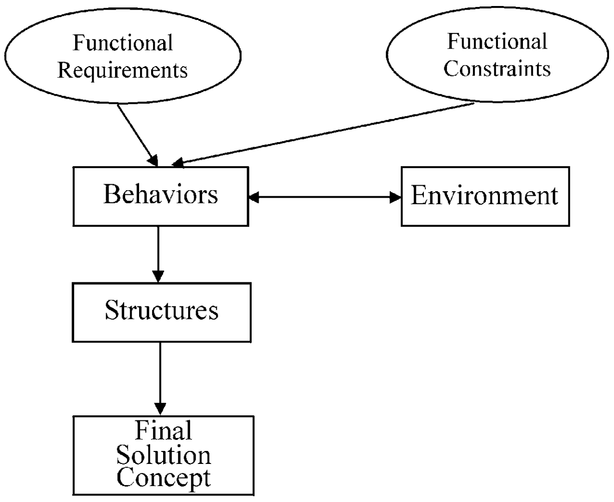


Fig. 1. Basic flow of functional design process.

In EFDEX, we have extended Deng's [9] dual-step Function-Environment-Behaviour-Structure (FEBS) modeling framework, which was initially developed for an interactive functional design environment, to develop our proposed Extended FEBS modeling framework, which will be suitable for an intelligent functional design environment. The initial dual-step FEBS model has involved diverse design characteristics (function, behavior, structure and environment) in a unified design environment, and proved to be suitable for an interactive functional design environment. However, when applied in an intelligent functional design environment, its dual-step modeling strategy which means the first step (initial function decomposition and conversion) and the second step (causal behavioral process generation) is inflexible due to its fixed up-down sequence. Any problem encountered in the second step has to be solved within that level, and cannot be returned to the previous level for further function decomposition. Hence, we develop an Extended FEBS modeling framework to solve this problem.

In our proposed Extended FEBS model, the interrelationships among various design characteristics will be elaborated and stressed. Behaviour is mainly represented in terms of driving input, functional output and side effect, as adapted from the initial dual-step FEBS modeling framework [9]. Before analyzing it, we adopt the notation specified by Kusiak and Szczerbicki [10], which is shown in Figure 2.

According to this notation, Fig. 3 shows an illustrative example of the Extended FEBS modeling framework, which consists of three layers: the function, behavior and environment layers. A continuous directed line indicates an internal relationship in the same layer. A dashed directed line means an external relationship between different layers.

The highest layer is a function layer, in which the overall functional requirement is usually decomposed into sub-functions hierarchically in such a way that its representation gradually becomes concrete and fine. For example, function *F1* is decomposed into sub-functions *F11* and *F12*; function *F11* is decomposed into its sub-functions *F111*, *F112* and *F113*. However, one problem is at which level the function decomposition should be carried out, or what the starting or stopping points of the

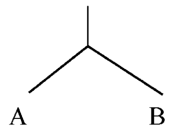
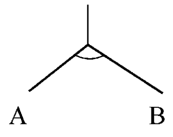
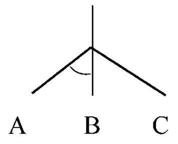
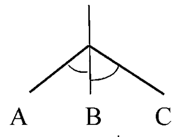
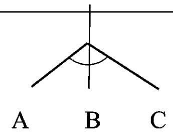
AND/OR clause representation	
Graphical	Literal
	A OR B
	A AND B
	A AND B OR C
	A AND B OR B AND C
	A AND B AND C

Fig. 2. Graphical and literal representation of AND/OR clauses [10] Copyright © 1990 by ASME.

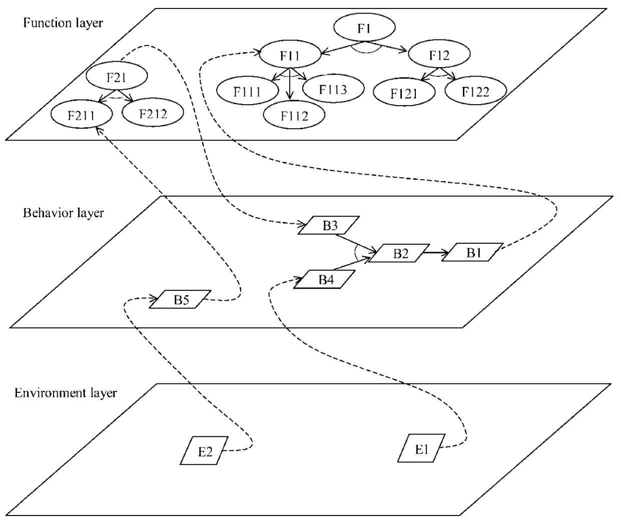


Fig. 3. Extended FEBS functional modeling framework.

decomposition should be. The relevant problem solving strategy will be illustrated in Section 5.3 with the aid of knowledge engineering.

The middle layer is a behavior layer. Because structure is the concrete spatial configuration of behavior, if the behavioral configuration of an overall design object is acquired, the structural configuration is also determined. Hence, structure is actually involved in the behavior layer implicitly. In the behavior layer, a set of behaviors is interconnected with each other, with one's functional output achieving the other's functional requirement (e.g. driving input), so as to achieve the external function in the function layer. Of course, all the connected behaviors must stringently satisfy the imposed functional constraints. For example, behavior *B3*'s functional output and behavior *B4*'s functional output can mutually achieve behavior *B2*'s functional requirement (e.g. driving input), while behavior *B2*'s functional output can achieve behavior *B1*'s driving input. Finally, behavior *B1*'s functional output achieves the external function *F11* in the function layer. In summary, function *F11* is achieved by behavior *B1*, whose functional requirement is achieved by its interconnected behaviors. In addition, all these selected behaviors, such as *B1*, *B2*, *B3* and *B4*, must stringently satisfy the imposed functional constraints.

We can see that there is a tight relationship between function and behavior. For example, function *F11* in the function layer can be achieved by behavior *B1* in the behavior layer. The driving input of behavior *B3* in the behavior layer can be projected onto the function layer to be functional requirement *F21*, which can be broken down into sub-functions *F211* and *F212* by means of function decomposition. However, one problem is when a function in the function layer should be fulfilled by a behavior in the behavior layer, and when the functional requirement of a behavior in the behavior layer should be projected onto the function layer for further function decomposition. The relevant problem solving strategy will be illustrated in Section 5.3 with the aid of knowledge engineering.

The lowest layer is an environment layer. A working environment can enable the functional output to achieve the functional requirement of behavior in the behavior layer. For example, the functional output of environment *E1* in the environment layer provides the driving input of behavior *B4* in the behavior layer. The point at which to match the functional requirement of behavior in the behavior layer to the functional output of the environment in the environment layer will be discussed in Section 5.3 with the aid of knowledge engineering.

In this Extended FEBS modeling framework, function, behavior and environment are respectively modeled in three layers with hierarchical, interactive and causal relationships, and its modeling strategy is in a flexible, two-way mode which is quite different from that of the initial dual-step FEBS model [9]. For example, a function in the function layer can be achieved by a behavior in the behavior layer, as is the transformation from the function layer to the behavior layer; on the other hand, a complicated functional requirement of a behavior in the behavior layer can be projected onto the function layer to be broken down into sub-functions, as is the transformation from the behavior layer to the function layer. The flexible modeling strategy of this Extended FEBS model will be quite useful in developing an intelligent functional design environment.

In the Extended FEBS modeling framework, we have developed the functional relationships among function, behavior, constraint, structure and environment, especially the hierarchical and causal relationships among function, behavior and environment. Based on this flexible, causal and hierarchical modeling framework, an intelligent functional reasoning process can be performed successfully with the aid of knowledge engineering, as will be introduced in Section 5.3.

4. Knowledge-Based Functional Representation

As an important branch of AI, knowledge-based techniques for the function-oriented upstream stage of the design process have been an active area of research for the past decade, and many research papers have been published. In the functional design domain, a knowledge-based system is mainly used to solve modeling and reasoning problems. The most common forms of knowledge representation include rules, frames and objects. The rule-based paradigm was adopted by Li et al. [11] to automate the computational synthesis of the conceptual design of mechanisms. The design algorithm employs best-first heuristic searches in a library of mechanical devices, represented and classified qualitatively. Moulianitis et al. [12] presented a rule-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 finds a final solution from a given set of feasible solutions, and can synthesize new solutions to accomplish the required specifications. Besides rule-based representation, frame-based representation is also widely used. Tong and Gomory [13] used a frame-based structure to model parts of standard kitchen appliances. They use a goal-driven strategy. An increasingly popular modeling representation is the object-oriented representation. Akagi and Fujita [14] used an object-oriented architecture for supporting the functional design process. The model for the design process is constructed using networks composed of knowledge elements, which are represented modularly in objects. This modeling results in the determination of design variables flexibly in the conceptual design process. Gorti et al. [15] developed an object-oriented representation for product and design processes, with which design operators update and transform design contexts, and support a range of design automation, extending from manual design all the way to automated design (with various reasoning mechanisms, e.g. inheritance, rules, constraints and so on).

In our research project, as a prelude to a knowledge-based functional design system, we propose a knowledge-based functional representation scheme, in which rule-based and object-oriented knowledge representation are integrated to represent the function-related design characteristics in an intelligent design environment.

4.1. Rule-Based Knowledge Representation

A rule-based system allows the 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. In EFDEX, the rule paradigm is very important in performing the intelligent functional reasoning process. Some rules in the rule base can be treated as a rule subset to be attached to an independent entity or design object. The attached rule subset can control the design procedure directly relating to this design object, and expresses causal knowledge involving different design objects. This mechanism also allows the design object to be defined without repetition of a basic rule subset. How the rule subset is attached to a design object will be discussed in Section 4.2. In this section, we present two kinds of production rules classified into domain-specific and general rules.

4.1.1. Domain-Specific Rules

Domain-specific rules refer to a set of rules that are used to solve domain dependent problems. In

EFDEX, we have used the proposed methodology to perform a functional design of an automatic assembly system for manufacturing electronic connectors. In this application domain, 255 domain-specific rules have been developed, some of which are formulated in Table 1.

In the case that a desired domain-specific function (overall functional requirement or the functional requirement of any retrieved behavior) is too complicated to be matched with the functional output of any behavior, the desired function should be broken down into fewer complex sub-functions by means of some domain-specific function decomposition rules, like the rules mentioned in Table 1, to facilitate the subsequent search for causal matching behaviors.

4.1.2. General Rules

General rules refer to a set of rules that are used to solve general problems. EFDEX can be applied to various application domains after changing the domain-specific behavior base and domain-specific rules. We formulate the examples of some general production rules as follows:

```
Rule Anti_Loop
  IF      last matching behavior object found
          belongs to a previously selected list of
          present searching branch
  THEN    discard this searching branch
  AND     start searching next branch
```

This is an anti-looping rule, without which some behavior objects may be called recursively. This rule is applied each time a behavior object is found.

```
Rule Search_Branch_End
  IF      all driving inputs of behavior objects are
          available in environment
  AND     side effects of behavior objects are suc-
          cessfully prevented
  THEN    terminate this branch
  AND     start searching next branch
```

This rule is used to a terminate a searching branch and put it in the configuration list. This rule is applied each time a behavior object is retrieved to working memory.

4.2. Knowledge-Based Functional Representation Scheme

4.2.1. Knowledge-Based Functional Representation Scheme and Object-Oriented Behaviour Base

In Section 2, we described various design characteristics related to functional design (i.e. function, behavior, structure, environment and constraint). In Section 4.1, we presented production rules whose subsets can be attached to the intelligent functional objects. In this section, we present the knowledge-based functional representation scheme, in which behaviors are defined as intelligent functional objects

Table 1. Examples of domain-specific rules

Rule Name	IF (condition)	THEN (conclusion)
Decompose1	a desired function is *Insert terminal into housing*	decompose it into *Clamp housing after locating housing* AND *Insert terminal after holding terminal*
Decompose2	a desired function is *Clamp housing after locating housing*	decompose it into *Locate housing* AND *Clamp housing*
Decompose3	a desired function is *Insert terminal after holding terminal*	decompose it into *Hold terminal* AND *Insert terminal*
Decompose4	a desired function is *Transmit torque*	decompose it into *Apply torque* AND *Receive torque*
Decompose5	a desired function is *Provide translational motion with a certain range*	decompose it into *Provide translational motion* AND *Control moving range*
Decompose6	a desired function is *Compensate offset* AND precision constraint is *high*	decompose it into *Compensate radial offset by applying a sliding element* AND *Compensate axial offset by applying a sliding element*
Decompose7	a desired function is *Compensate offset* AND precision constraint is not *high*	decompose it into *Compensate radial offset without using a sliding element* AND *Compensate axial offset without using a sliding element*
Decompose8	a desired function is *Bend terminal after holding terminal*	decompose it into *Hold terminal* AND *Bend terminal*

encompassing various design characteristics and rule subsets, and saved in the object-oriented behavior base.

Here the object-oriented behavior representation we originally presented [16] is refined, and the rule subset and functional constraint satisfaction attributes are encapsulated explicitly in the behavior object. These important enhancements will enable us to successfully implement the methodology in a computer program, which can perform functional design more intelligently.

Object-orientation is usually both a language feature and a design methodology [17]. An object is an entity that combines its data structure and its methods into one. We define a behavior object as an intelligent functional object. The most generic behaviors can be represented as the top-most generic class object. This class object is defined as follows:

```
Class Behaviour {
  Basic design attributes:
    Name:
    Structure:
    Driving_Input:
    Functional_Output:
    . . . . . . . .
  Functional constraint satisfaction attributes:
    Precision:
    . . . . . . . .
  Rule subset:
    Anti_Loop;
    Search_Branch_End;
    . . . . . . . .
  Methods:
    Input_Data ( );
    Output_Data ( );
    . . . . . . . .
}
```

As shown in the above pseudo-code, the generic class *Behaviour* encapsulates basic design attributes, functional constraint satisfaction attributes, the rule subset and methods all of which will be illustrated in detail in the following.

The basic design attributes *Name*, *Structure*, *Driving_Input* and *Functional_Output* are (respectively) the name, meeting structure, driving input and functional output of defined behavior. Because function is embodied with behavior, and behavior is met with structure, we encapsulate the driving input (a kind of functional requirement), functional output and structure in the behavior object. Note that the design attribute *Structure* does not mean that only the structure name can be incorporated. In fact, it is the schematic geometric structure. Figure 4 shows the schematic geometric structures of some behaviors in a case-specific domain. *Driving_Input* and *Functional_Output* are the causal attributes of behavior. They are specific kinds of functional terms, which can be manipulated by means of a functional vocabulary library introduced in Section 2.2. They are very useful for intelligent causal reasoning techniques, which automatically search for an unknown behavior object based on the known functional description or known behavior object.

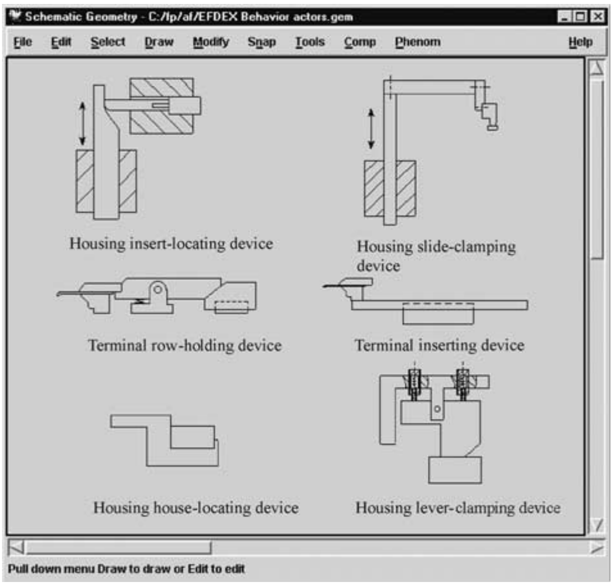

Schematic Geometry - C:/lp/al/EFDEX Behavior actors.gem
File Edit Select Draw Modify Snap Tools Comp Phenom Help
Housing insert-locating device
Housing slide-clamping device
Terminal row-holding device
Terminal inserting device
Housing house-locating device
Housing lever-clamping device
Pull down menu Draw to draw or Edit to edit

Fig. 4. Schematic geometric structures of some behaviours.

The functional constraint satisfaction attributes (e.g. *Precision*) are the attributes to satisfy relevant functional constraints. *Precision* can be assigned with three alternative concrete values: *high*, *middle* and *low*. It will be an unsuccessful functional design only to achieve functional requirements without considering functional constraints simultaneously.

The rule subset includes some attached rules, such as *Anti_Loop* and *Search_Branch_End*, which have been introduced in Section 4.1.2 and are (respectively) used for anti-looping and ending the search branch. The attached rule subset can control the relevant design procedure relating to this behavior object.

The methods such as *Input_Data ( )* and *Output_Data ( )* are also encapsulated as procedural programs. This mechanism allows the behavior objects to be defined without repetition of some basic procedural programs. The method *Input_Data ( )* will help the user to input the required concrete values to all of the design attributes. The method *Output_Data ( )* will return the relevant values of design attributes to the working memory.

4.2.2. Inheritance Property of Knowledge-Based Functional Representation Scheme

The inheritance mechanism from object-oriented technology makes the data structure of this representation tight, concise and easy to manipulate, thus making it possible to avoid any repetition of common data. With this mechanism, the other kind of behavior class, which includes the extra basic design attribute *Side_Effect*, can be represented as the child class object of the generic class *Behaviour*. It can inherit the latter's attributes, rule subset and methods, and also add specific attributes, rule subset or methods pertinent to itself. For example, the child class can be represented as:

```
Class Behaviour_With_Side_Effect
{
    Inherit: Behaviour
  Basic design attributes:
    Side_Effect:
    .......
  Method:
    Prevent_Side_Effect ( );
    .......
}
```

Here, the basic design attribute *Side_Effect* defines the side effect of the defined behavior object. It is a specific kind of functional term which can be manipulated by means of the functional vocabulary library, introduced in Section 2.2. The method *Prevent_Side_Effect ( )* can prevent this side effect by searching for and retrieving another behavior object whose functional output can prevent the former behavior's side effect.

A good functional design system should support seamless transformation from the functional design phase to the detailed design phase. Our proposed knowledge-based functional representation scheme can facilitate the capture of both abstract and detailed knowledge, and allows mapping from one kind of knowledge to another. This is because all of the detailed design-related data such as various parameters (properties) of function-related design characteristics can also be incorporated in this representation scheme. In the generic behavior class object (introduced in Section 4.2.1), the basic design attributes such as driving input, structure, and functional output, are only expressed qualitatively, exclusive of detailed design parameters which should be expressed quantitatively. For example, the *Functional output* of a mechanical device *Cylinder* is only expressed as *Provide translational motion*, while the detailed design parameters such as speed or load are not illustrated. Owing to the inheritance mechanism and object structure, a lower-level detailed-behavior class object which encapsulates all the quantitative detailed design parameters, and also inherits the qualitative design attributes from the functional design phase, can be produced to aid in the seamless integration between the functional design phase and the detailed design phase. Its concrete representation scheme is ignored in this paper.

The above behavior objects are represented by class. A class is only a template for the structure of objects, which only encapsulates attributes, rule subset and methods, not concrete data. We can instantiate a behavior class by assigning concrete data to its attributes, to result in a behavior instance. When a new behavior object is added to the object-oriented behavior base, it is compared with the objects that are already in the behavior base. If it is the same as a behavior 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, while the similar parts are derived from the class of the higher level by inheritance. Thus, object-oriented representation is very convenient for maintaining and modifying the behavior base.

5. Architecture of EFDEX

We have proposed an Extended FEBS modeling framework in Section 3 to build the functional relationships among function, behavior, constraint, structure and environment, and a knowledge-based functional representation scheme in Section 4 to allow intelligent communication with the computer software. On the basis of these, we now introduce the EFDEX expert system.

5.1. Overview of EFDEX

There are three main components that constitute EFDEX: a user interface, a knowledge base and an inference engine:

1. The user interacts with EFDEX through a user interface. The design of the user interface employs user-friendly features such as 'question-and-answer', being menu-driven and with a GUI to guide the user throughout the consultation session.
2. The heart of EFDEX is a hybrid knowledge base which integrates the rule base and object-oriented

behavior base effectively. Rules can be used to express causal knowledge involving one or several design objects, and objects can also encapsulate rules. How we perform the integration of the object-oriented behavior base and rule base in a hybrid knowledge base has already been introduced in Section 4.

3. The inference engine is the interpreter for the knowledge base. The problems under examination during functional design involve both goal- and data-driven processes, and therefore backward chaining and forward chaining inference are used. In the section that follows, we introduce the inference engine in detail.

5.2. Formalizing the Knowledge

Knowledge formalization refers to refining the functional design knowledge into a form suitable for direct implementation in a computer system. In the present work, CLIPS [1], from the Software Technology Branch at the NASA/Lyndon B. Johnson Space Center, was used as the development tool.

Other than just 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). Hence, three modes of knowledge representation are available in CLIPS. These features are suitable for our proposed prototype knowledge-based system.

Take the domain-specific function decomposition rule *Decompose2*, introduced in Section 4.1.1, as an example. Below is its source code in CLIPS:

```
(defrule Decompose2                              Line 1
(declare (salience 50))                          Line 2
?parent-function<-(function
   (Verb clamp)(Noun housing)
   (Complement after locating housing)           Line 3
   (Matched no)(Decomposed no))
=>
(modify ?parent-function                         Line 4
   (Decomposed yes))
(assert (function (Verb locate)
   (Noun housing)
   (Complement NULL)(Matched no)
   (Decomposed no)))                             Line 5
(assert (function (Verb clamp)
   (Noun housing)
   (Complement NULL)(Matched no)
   (Decomposed no))))                            Line 6
```

Line 1 is the rule name. In line 2, a salience of 50 is declared in this rule. This feature is used to set the order of rule activation and execution. Line 3 denotes the left-hand side (condition) of the rule, and lines 4, 5 and 6 denote the right-hand side (conclusion) of the rule. Line 3 shows that a desired function *clamp housing after locating housing* is needed to fire the rule. Refer to Section 2.2: a function is formulated as '*Verb* + *Noun* + *Complement*'. In this left-hand side of the rule, *Verb* = *clamp*, *Noun* = *housing*, *Complement* = *after locating housing*. *Verb* and *Noun* are single-slots, while *Complement* is a multi-slot. To fire the rule, the desired function should also be one that has not yet been matched by a behavior object, or decomposed. Line 4 shows that the desired function has already been decomposed. Lines 5 and 6 are respectively the generated sub-functions *locate housing* and *clamp housing*. Similar explanation as for line 3 is given for lines 5 and 6, except that these two sub-functions have no complement.

5.3. Inference Engine and Intelligent Functional Reasoning Strategy

The inference engine applies the knowledge in the knowledge base to the case-specific data to arrive at some solution or conclusion. With the inference engine, Deng's [9] Causal Behavioral Process (CBP) reasoning framework, which was initially developed for interactive functional design, was extended by us to implement the intelligent causal behavioral reasoning strategy aiming at intelligent functional design. In EFDEX, the intelligent functional reasoning strategy of the inference engine includes data-driven and goal-driven approaches. The procedures that implement the control cycle are separated from the knowledge base. During analysis of the intelligent functional reasoning strategy, we adopt the notation shown in Fig. 2.

Figure 5 illustrates an example of the intelligent functional reasoning strategy of EFDEX. Recall from Section 4.1 that there are some domain-specific rules and general rules for knowledge representation, some of which are quoted here. The starting point of the inferencing strategy is to put the overall functional requirement *F1* into the working memory, and scan the behavior base to seek the behavior whose functional output can match functional requirement *F1*. Of course, all the retrieved behaviors must stringently satisfy the imposed functional constraints, and this statement will not be repeatedly described in the following discussions. In practice, functional constraints should be considered together with functional requirements.

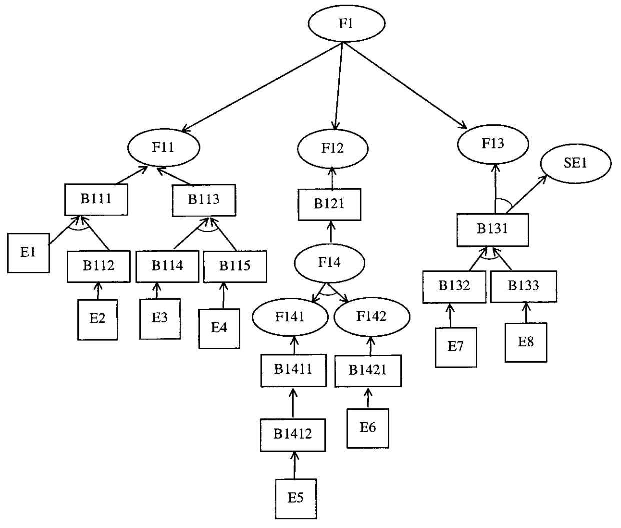


Fig. 5. Search tree of EFDEX.

If no matching is found after scanning the entire behavior base, then the inference engine scans the rule base to search for the problem solving production rules so as to break down the desired function into less complex sub-functions. Now if the premise of one domain-specific function decomposition rule is satisfied by overall functional requirement *F1*, then this rule will be fired, and its conclusion puts functions *F11*, *F12* and *F13* in the working memory as sub-functions. Here, the search strategy for function decomposition employs a data-driven control regime.

For sub-function *F11*, the starting point of the inferencing strategy is to scan the behavior base to search for the matching behavior whose functional output can achieve this desired sub-function *F11*. It is supposed that either functional output of behavior *B111* and *B113* can achieve *F11*. Then behaviors *B111* and *B113* are retrieved to the working memory, and their driving inputs are taken to be the new functional requirements. Suppose that behavior *B111* has two driving inputs: *Driving_Input1* and *Driving_Input2*. Because *Driving_Input1* is available in environment *E1*, the general rule *Search_Branch-_End* is fired to terminate this branch. Supposing that the functional output of behavior *B112* can match the *Driving_Input2* of behavior *B111* when the inference engine continues to scan the behavior base, behavior *B112* is also retrieved to the working memory. Because the driving input of behavior *B112* is available in environment *E2*, the general rule *Search_Branch_End* is fired to terminate this branch. This search strategy for the causal behavioral reasoning process employs a goal-driven control regime. The strategy retrieves *B111* and *B112* to achieve *F11*. Similarly, behaviors *B113*, *B114* and *B115* could also be found to achieve *F11*.

Similarly, it is supposed that behavior *B121* is retrieved to achieve sub-function *F12*, and its driving input is now taken to be the new functional requirement. The inference engine continues to scan the behavior base to seek the behavior whose functional output can match the driving input of behavior *B121*. It is supposed that no matching is found after scanning the entire behavior base. Then the inference engine scans the problem solving production rule base to search for the matching rule. Suppose that one domain-specific function decomposition rule is fired, and its premise is exactly the driving input (functional requirement *F14*) of *B121*, and its conclusions are sub-functions *F141* and *F142*. The process continues, and behaviors *B1411* and *B1412* are developed to achieve sub-function *F141*, and behavior *B1421* is developed to achieve sub-function *F142*.

Similarly, it is supposed that the behavior *B131* is retrieved to achieve sub-function *F13*, and its driving input is now taken to be the new functional requirement. The process continues, and behavior *B132*, whose functional output can match the driving input of behavior *B131*, is developed. However, besides providing a functional output, behavior *B131* also produces an unexpected side effect, *SE1*. Then the new functional requirement to prevent side effect *SE1* of behavior *B131* needs to be achieved. Thus, the inference engine continues to scan the behavior base to seek the behavior whose functional output can prevent side effect *SE1* of behavior *B131*, and retrieves the matching behavior *B133*.

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

Variant #1→ (*B111* + *B112*) + (*B121* + *B1411* + *B1412* + *B1421*) + (*B131* + *B132* + *B133*)
Variant #2→ (*B113* + *B114* + *B115*) + (*B121* + *B1411* + *B1412* + *B1421*) + (*B131* + *B132* + *B133*)

Figure 6 shows both behavioral configurations of the above illustrated potential concept variants. Both can achieve overall functional requirement *F1* and the imposed functional constraints.

In the above intelligent functional reasoning strategy, the emphasis is that the system always searches for a matching behavior in the object-oriented behavior base, whose functional output matches the desired function, and that also stringently satisfies the functional constraints as a starting point. When no matches exist, the system allows for the automatic decomposition of the desired function into sub-functions, by means of relevant domain-specific function decomposition rules in the rule base. The search strategy reduces the possibility of combinatorial explosion, that can occur during function decomposition.

6. Case Study

In this section, we study one design case so as to test the applicability of EFDEX. During our illustration, the intelligent functional design process of the automatic assembly system for manufacturing electronic connectors will be studied. The automatic assembly system comprises of a vibrator bowl feeding unit, housing singulator, walking beam unit, terminal inserting unit, terminal cutting unit, terminal bending unit, and so on. Among them, the terminal inserting unit is the main and most complicated unit,

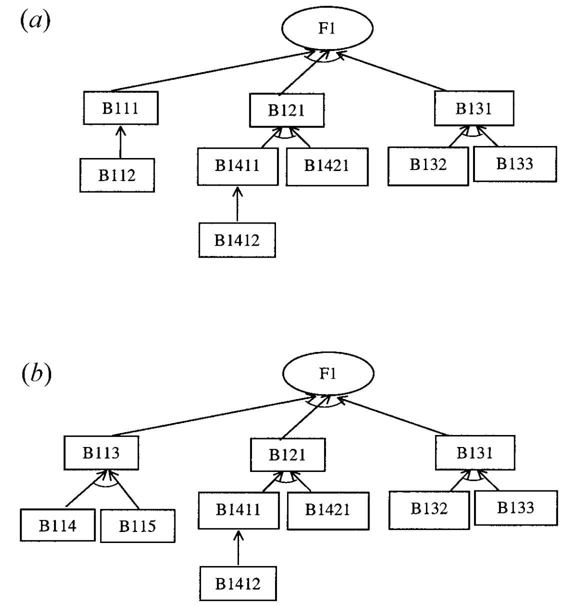


Fig. 6. Behavioural configuration of potential concept variants.

compared with the other units, so this case study will focus on the functional design for the terminal inserting unit of this automatic assembly system.

6.1. Problem Description and User Input

Suppose the following design specifications are given:

1. Design the terminal inserting unit whose overall functional requirement 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 functional constraint applies: *The inserting position tolerance* $< 0.1mm$. (which means *High precision is needed*).

6.2. Intelligent Functional Reasoning Process and System Output

Recall from Section 4.1 that there are some domain-specific rules and general rules for knowledge representation, some of which are quoted here. Referring to Fig. 7, the logical steps of the inferencing strategy are as follows:

1. The starting point of the inferencing strategy is to put the overall functional requirement *F1* in the working memory, and scan the behavior base to seek the behavior whose functional output can match the overall functional requirement *F1*. Of course, all the retrieved behaviors must stringently satisfy the imposed functional constraint: *high precision* (in other words, the value of the functional constraint satisfaction attribute *Precision* should be *high* for all the retrieved behaviors), and this statement will not be repeatedly described in the following discussions. Supposing that no matching is found after scanning the whole behavior base; the inference engine starts to scan the rule base to search for the problem solving rules. Then with its desired function *F1* satisfied, domain-specific rule *Decompose1* is fired to decompose function *F1* into sub-functions *F11* and *F12*, where:

   F1: *Insert terminal into housing*;
   F11: *Clamp housing after locating housing*;
   F12: *Insert terminal after holding terminal.*

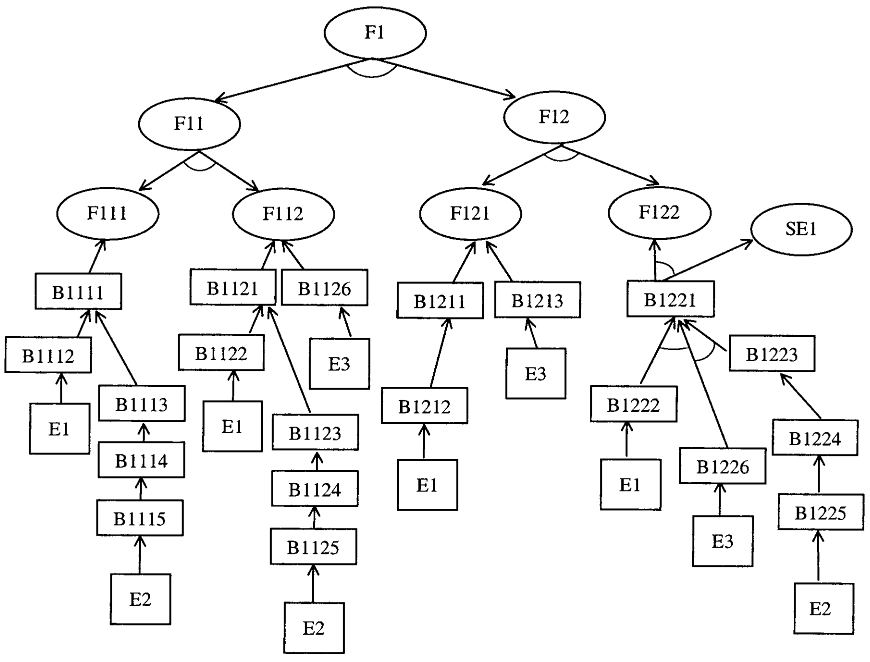


Fig. 7. Search tree of EFDEX in case study.

2. Similarly, *F11* is decomposed into *F111* and *F112* with rule *Decompose2* activated, where:
   F111: *Locate housing*;
   F112: *Clamp housing*.
3. For *F111*, the starting point of the inferencing strategy is to scan the behavior base to search for the matching behavior whose functional output can match the desired sub-function *F111*. It is found that either behaviors *B1111* or *B1116* can achieve *F111*, but *B1116* is discarded because its value of the functional constraint satisfaction attribute *Precision* is *low*, and it can not satisfy the imposed functional constraint. Then, only behavior *B1111*, which also satisfies the functional constraint *High precision*, is retrieved to the working memory, and its driving input is taken to be the new functional requirement. Similarly, behavior *B1112* is developed with its functional output *Provide translational motion* matching behavior *B1111*'s driving input *Provide translational motion*. Now *B1112*'s driving input *Provide pneumatic air* becomes the new functional requirement. Because 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 insert-locating device*;
   B1112: *Cylinder device*;
   B1116: *Housing house-locating device*.
4. Similarly, the alternative causal behavioral branch to achieve the driving input of behavior *B1111* can be developed, where:
   B1113: *Cam device*;
   B1114: *Gear pair device*;
   B1115: *Motor device*.
   Now the causal behavioral searching process for realising function *F111*: *Locate housing* has been finished with two feasible branches developed.
5. The alternative causal behavioral branches for achieving function *F112* can be developed, where:
   B1121: *Housing slide-clamping device*;
   B1122: *Cylinder device*;
   B1123: *Cam device*;
   B1124: *Gear pair device*;
   B1125: *Motor device*;
   B1126: *Housing lever-clamping device*;
6. *F12* can be decomposed into *F121* and *F122* with rule *Decompose3* activated, where:
   F121: *Hold terminal*;
   F122: *Insert terminal*.
7. The causal behavioral branches for *F121* and *F122* can be developed, where:
   B1211: *Terminal row-holding device*;
   B1212: *Cylinder device*;
   B1213: *Terminal side-holding device*;
   B1221: *Terminal inserting device*;
   B1222: *Cylinder device*;

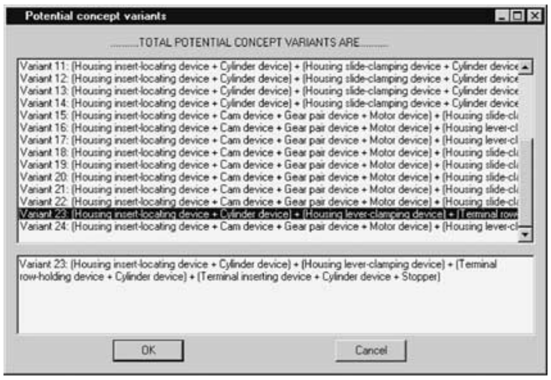
B1223: *Cam device*;
B1224: *Gear pair device*;
B1225: *Motor device*.
B1226: *Stopper*.

The explanation for the side effect *SE1*: *Terminal moves too much* is noted below: The behavior *B1221* is developed to achieve the function *F122*, but the behavior *B1221* simultaneously produces the side effect *SE1*, which should be prevented. So the inference engine scans the behavior base to search for the behavior whose functional output can prevent behavior *B1221*'s side effect *SE1*. Then behavior *B1226* is retrieved with its functional output being *Prevent terminal moving too much*.

8. 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 is shown in Fig. 8.
9. According to Pahl and Beitz [18], we evaluate all the resulting concept variants to narrow the choice. This final decision-making phase is the phase of concept evaluation and selection, where all the concept variants generated are evaluated based on technical and economic criteria, and the highest scoring variant is selected as the best concept variant. In this example, Concept variant #23 can be chosen as the final solution concept, and all its retrieved interconnected behaviors are shown as follows:

Variant 23: (*Housing insert-locating device* + *Cylinder device*) + (*Housing lever-clamping device*) + (*Terminal row-holding device* + *Cylinder device*) + (*Terminal inserting device* + *Cylinder device* + *Stopper*)

Figure 9 shows the graphical representation of this final solution concept.

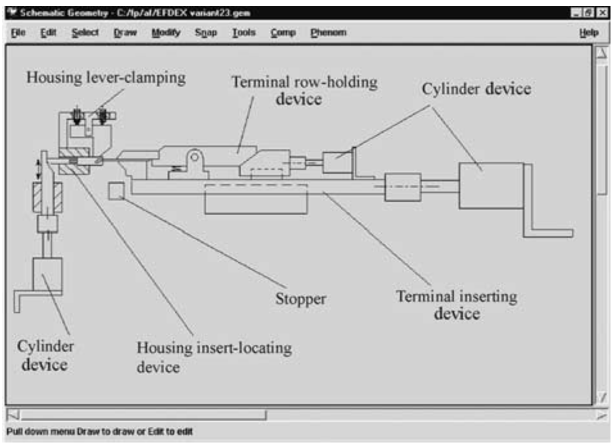


......TOTAL POTENTIAL CONCEPT VARIANTS ARE......

```
Variant 11: (Housing insert-locating device + Cylinder device) + (Housing slide-clamping device + Cylinder device
Variant 12: (Housing insert-locating device + Cylinder device) + (Housing slide-clamping device + Cylinder device
Variant 13: (Housing insert-locating device + Cylinder device) + (Housing slide-clamping device + Cylinder device
Variant 14: (Housing insert-locating device + Cylinder device) + (Housing slide-clamping device + Cylinder device
Variant 15: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 16: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing lever-cl
Variant 17: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing lever-cl
Variant 18: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 19: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 20: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 21: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 22: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing slide-cl
Variant 23: (Housing insert-locating device + Cylinder device) + (Housing lever-clamping device) + (Terminal row
Variant 24: (Housing insert-locating device + Cam device + Gear pair device + Motor device) + (Housing lever-cl
```

```
Variant 23: (Housing insert-locating device + Cylinder device) + (Housing lever-clamping device) + (Terminal
row-holding device + Cylinder device) + (Terminal inserting device + Cylinder device + Stopper)
```

Fig. 8. User interface for potential concept variants.


Schematic Geometry - C:/lp/al/EFDEX variant23.gem
File
Edit
Select
Draw
Modify
Snap
Tools
Comp
Phenom
Help
Housing lever-clamping
Terminal row-holding device
Cylinder device
Stopper
Terminal inserting device
Cylinder device
Housing insert-locating device

Pull down menu Draw to draw or Edit to edit

Fig. 9. Graphical representation of concept variant #23 for terminal inserting unit.

7. Related Research

Functional reasoning as a design approach has been around for at least 30 years now, and has attempted to support design in the conceptual stage by methods and approaches to describe function, to establish function structures, to satisfy these sub-functions and combine them into concept alternatives, and to evaluate them [19]. In recent years, behavior representation has formed the foundation of various functional reasoning techniques [20]. A behavioral reasoning model linking function and structure has been presented by Gero et al. [21].

In the context of motion transmission mechanism design, Hoover and Rinderle [22] adopted a behavior-assisted synthesis strategy by transforming design specifications in a function-preserving manner to obtain function structures which correspond closely to collections of available components. Chakrabarti and Bligh [23–25] suggested a functional synthesis approach using a set of functional elements (primitives) as building blocks. The synthesis process involves finding various transformations of the input-output characteristics of flow variables (kind, orientation, sense, position and magnitude) for a set of basic structures, and combining them to find transformations which are the requirements of a particular design problem. The proposed input-output synthesis approach can be regarded as a kind of behavior-assisted reasoning approach, because an input-output flow is in effect a kind of system behavior. Li et al. [26] used the configuration space as the basis on which to represent the behaviors of kinematic pairs, which are the basic building blocks from which more complex design is synthesised. Design solutions that consist of either a chain or a network of multiple kinematic pairs can be generated.

Some other researchers use a bond graph approach to deal specifically with the flow of energy. Bond graphs provides a convenient and uniform representation of the dynamic behavior of a broad class of physical systems. Ulrich and Seering [27] used a bond graph for the synthesis of schematic descriptions in mechanical design. Bracewell and Sharpe [5] developed a conceptual design environment called *Schemebuilder* using bond graphs. Rules for functional decomposition have been provided for assisting functional reasoning, based on the bond graph principles for the embodiment of energetic systems. Welch and Dixon [28] developed behavior graphs (derived from bond graphs) to facilitate behavioral reasoning, so that the functional requirements can be transformed into a behavioral representation, which is then further used to guide the early stage of embodiment design.

Two streams of research work have influenced us most: the Function-Behaviour-State (FBS) modeler developed by Umeda et al. [4, 29], and Deng's [9] dual-step Function-Environment-Behaviour-Structure (FEBS) modeling framework. The FBS modeler uses physical state transition and physical phenomena to represent the behavioral process. It reasons about function by means of two approaches: causal and task decomposition. With this distinction, they clarify that the decomposition knowledge in the function prototype includes only the task decomposition. Causal decomposition of function is used in reasoning out candidates of additional physical features that are to satisfy the conditions for the physical features identified from the function-behavior relationship. Deng [9] argued that causal decomposition of function should be more appropriately extended by the behavioral process generation task. In addition, a product can only function in a certain (intended) working environment. Hence, Deng suggests a dual-step FEBS modeling procedure, consisting of initial function decomposition and causal behavioral process generation.

The knowledge-based functional reasoning strategy presented in this paper is based on the above function decomposition and causal behavioral process, but differs from previous research in the following ways:

1. Functional reasoning is based on a flexible, causal and hierarchical functional modeling framework (Extended FEBS modeling framework), whose modeling strategy is in flexible, two-way mode, which is quite different from that of the initial dual-step FEBS model [9]. Environment is represented explicitly in an environment layer to provide the functional output, which then achieves the functional requirement of behavior in the behavior layer. This enhancement will facilitate developing a complete and flexible functional design model.
2. Given a desired function, the inference engine first scans the object-oriented behavior base to search for a behavior with a functional output that matches the desired function. Only if no matching behavior can be found will the desired function then be automatically decomposed into sub-functions, using a domain-specific function decomposition rule. This search strategy differs from most other works [4, 9, 29], in that function decomposition is not used as a starting point of a functional reasoning strategy, hence reducing the possibility of combinatorial explosion that can occur during function decomposition.

8. Conclusion

This paper describes our proposed system EFDEX, which applies a knowledge-based approach to the functional design of engineering systems so that functional design can be performed intelligently. We have developed an Extended FEBS functional modeling framework in which the functional relationships among function, behavior, constraint, structure and environment (especially the causal and hierarchical relationships among function, behavior and environment) are developed. With this flexible, hierarchical and causal functional modeling framework, our proposed knowledge-based functional reasoning strategy can automatically reason out physical behavior from a desired function or desired behavior. In addition, complicated desired functions which cannot be matched with the functional output of any behavior after searching the object-oriented behavior base, will automatically be decomposed into sub-functions by means of relevant function decomposition rules. In this paper, a knowledge-based functional representation scheme which integrates two popular AI representation techniques (object-oriented representation and rule-based representation) is also proposed, to allow intelligent communication with computer software.

EFDEX was developed using CLIPS, a declarative programming language. The work also demonstrates the potential of developing similar knowledge-based expert systems for other practical applications.

References

1. Giarratano, J., Riley, G. (1998) Expert Systems: Principles and Programming, 3rd ed. Boston, PWS
2. Schemekel, H. (1989) Functional models and design solutions. Annals of the CIRP, 38(1), 129–132
3. Fink, P. K., Lusth, J. C. (1987) Expert systems and diagnostic expertise in the mechanical and electrical domains. IEEE Transactions on System, Man and Cybernetics, 17(3), 340–349
4. Umeda, Y., Takeda, H., Tomiyama, T., Yoshikawa, H. (1990) Function, behavior and structure. In: Gero, J. S. (Editor), Applications of Artificial Intelligence in Engineering V, 177–193
5. Bracewell, R. H., Sharpe, J. E. E. (1996) Functional descriptions used in computer support for qualitative scheme generation – 'Schemebuilder'. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 10(4), 333–345
6. Deng, Y.-M., Britton, G. A., Tor, S. B. (1998) A design perspective of mechanical function and its object-oriented representation scheme. Engineering with Computers, 14, 309–320
7. Tor, S. B., Britton, G. A., Chandrashekar, M., Ng, K. W. (1998) Functional design. In: Usher, J., Utpal, R., Parsaei, H. (Editors), Integrated Product and Process Development: Methods, Tools and Technologies. New York, Wiley, 29–58
8. Prabhakar, S., Goel, A. (1998) Functional modeling for enabling adaptive design of devices for new environments. Artificial Intelligence in Engineering, 12, 417–444
9. Deng Y.-M. (2000) Functional Design of Mechanical Products: Design Model and Modeling Framework. PhD thesis, Nanyang Technological University, Singapore
10. Kusiak, A., Szczerbicki, E. (1990) A formal approach to design specifications. In: Ravani, B. (Editor), Advances in Design Automation, ASME, Vol. 1, 311–316
11. Li, C. L., Tan, S. T., Chan, K. W. (1996) A qualitative and heuristic approach to the conceptual design of mechanisms. Engineering Application of Artificial Intelligence, 9(1), 17–31
12. Moulianitis, V. C., Dentsoras, A. J., Aspragathos, N. A. (1999) A knowledge-based system for the conceptual design of grippers for handling fabrics. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 13, 13–25
13. Tong, C., Gomory, A. (1993) A knowledge based computer environment for the conceptual design of small electromechanical appliances. Computers, 26(1), 69–71
14. Akagi, S., Fujita, K. (1990) Building an expert system for engineering design based on the object-oriented knowledge representation concept. Journal of Mechanical Design, 112, 215–222
15. Gorti, S. R., Gupta, A., Kim, G. J., Sriram, R. D., Wong, A. (1998) An object-oriented representation for product and design process. Computer-Aided Design, 30(7), 489–501
16. Zhang, W. Y., Britton, G. A., Tor S. B. (2000) A prototype knowledge-based system for the conceptual synthesis of design process. International Journal of Advanced Manufacturing Technology, 17(8), 549–557
17. Meyer, B. (1988) Object-Oriented Software Construction. New York, Prentice-Hall
18. Pahl, G., Beitz, W. (1996) Engineering Design – A Systematic Approach. London, Springer-Verlag
19. Chakrabarti, A., Blessing, L. (1996) Special issue: representing functionality in design. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 10(4), 251–253
20. Umeda, Y., Tomiyama, T. (1997) Functional reasoning in design. IEEE Intelligent Systems & Their Applications, 12(2), 42–48
21. Gero, J. S., Lee, H. S., Tham, K. V. (1992) Behaviour: a link between function and structure in design. In: Intelligent CAD. Amsterdam, Elsevier, 193–220
22. Hoover, S. P., Rinderle, J. R. (1989) A synthesis strategy for mechanical devices. Research in Engineering Design, 1, 87–103
23. Chakrabarti, A., Bligh, T. P. (1994) An approach to functional synthesis of solutions in mechanical conceptual design. Part I: introduction and knowledge representation. Research in Engineering Design, 6(3), 127–141
24. Chakrabarti, A., Bligh, T. P. (1996) An approach to functional synthesis of solutions in mechanical conceptual design. Part II: kind synthesis. Research in Engineering Design, 8(1), 52–62
25. Chakrabarti, A., Bligh, T. P. (1996) An approach to functional synthesis of solutions in mechanical conceptual design. Part III: Spatial configuration. Research in Engineering Design, 2, 116–124
26. Li, C. L., Chan, K. W., Tan, S. T. (1999) A configuration space approach to the automatic design of multiple-state mechanical devices. Computer-Aided Design, 31(10), 621–653
27. Ulrich, K. T., Seering, W. P. (1980) Synthesis of schematic descriptions in mechanical design. Research in Engineering Design, 1, 3–18
28. Welch, R. V., Dixon, J. R. (1994) Guiding conceptual design through behavioral reasoning. Research in Engineering Design, 6, 169–188
29. Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., Tomiyama, T., (1996) Supporting conceptual design based on the function-behavior-state modeler. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 10(4), 275–288