# A Basis for Computer-Aided Generation of Design Concepts for Instrument Systems

by

Robert Patrick Ginger

A thesis submitted for the degree of Doctor of Philosophy

Measurement and Instrumentation Centre,
Department of Electrical, Electronic and Information Engineering,
City University, London EC1V OHB

September 1994

# CONTENTS

## CONTENTS

# LIST OF TABLES



# LIST OF FIGURES

## LIST OF FIGURES

## LIST OF FIGURES

# ACKNOWLEDGEMENTS

I would like to express my gratitude to the following people who have helped me with this work.

Professor L. Finkelstein, my supervisor during the entire course of this project. He has given invaluable support, encouragement and guidance when it was needed and his expertise and knowledge has provided the foundation on which this work is based.

Dr. M. El-hami, who also supervised me during the first two years, for his valued help in that time, also for his constructive criticism of earlier drafts of this thesis, and his assistance with the preparation of articles for publication.

Dr. S. Rozen, who during his stay at City University offered me his friendship and showed such enthusiasm for the subject. His advice has been of great assistance and has proved to be an excellent source of additional ideas.

I would also like to give my thanks to all other members of the staff in the Department of Electrical, Electronic and Information Engineering at City University who have given me their assistance and cooperation when required.

Last but not least, I would like to thank the Science and Engineering Research Council for their financial support, without which I could not have completed this study.

# DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

# ABSTRACT

Conceptual design is the phase of design where realisable solutions are found to the functional specification of a system. At present, this is not a well supported task owing to the creativity involved in it. This thesis describes a tool which provides automated support for one of the key activities in conceptual design, the proposal of candidate solutions for analysis and evaluation, in this case to the functional specification of an instrument system, a definition of a signal transformation to be realised. The thesis defines a classification of possible functions performed by an instrument system, the associated signal transformations and the parameters which influence the transformations. A functional specification of an instrument system and a corresponding solution can both be described in terms of a configuration of these functions and the constraints on the parameters of each function in the configuration. The system provides support for the generation of solutions to a functional specification defined using this representation and a functional specification defined at different levels of abstraction can be verified by an implementation of a model which represents the signal flow through a functional configuration. The support for solution generation offered by the system is based on a systematic search of different knowledge bases which contain information on existing solution characteristics, known functional configurations, and laws of physical effects. The methods of solution generation supported include use of existing solutions, use of first principles, systematic transformation of a functional specification, or use of analogy with the functional specifications of other similar solutions.

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation and Objectives

Recent developments in computing and artificial intelligence offer the potential for automated tools which aid the engineering design process. These tools are needed to facilitate cheaper design by making the design easier and also to help the designer cope with the rising complexity of the requirements for engineering systems. There are already many computer-aided design (CAD) systems for detailed design tasks, but fewer are available for the preliminary, more creative stages of design although in recent years the latter area of research has become an increasingly active one (Sriram and Adey, 1986; Gero, 1988; Mirza, Neves and Finkelstein, 1990; Gero 1991; Roosenburg, 1993). It is also the subject area of this thesis, which deals with computer-aided generation of conceptual design solutions for instrument systems, and in particular describes a knowledge based system (Michie, 1979; Hayes-Roth, Waterman and Lenat, 1983) which can be used as a design aid for this task.

A conceptual design solution is described in terms of the notion of a design concept, which is a subset of the set of all possible solutions to a problem. A design concept defines the solution space described by the constraints on the attributes of a solution or on the overall attributes of a combination of solutions. Conceptual design is a creative activity in which a designer will compare and contrast a number of different types of solution to the same problem, explore the effects of altering the boundaries of different solution spaces and think ahead to the detailed implications of any decisions taken. The purpose of the knowledge

based system to be described is to enable an instrument system designer to take a more creative approach to the exploration of solution spaces and hence to increase their awareness of how a problem can be solved. The solution spaces considered are those associated with systems which can be represented by specific types of models, namely lumped parameter, distributed parameter, and ray approximation system models.

## 1.2 Thesis Structure

The thesis is divided into two parts: the first part, which includes chapters two to four, is concerned with defining the task to be aided and the tools and techniques which can be used; the second part, chapters five and six, describes how the knowledge based system is to be implemented. It is organised into the following sequence of chapters:

Chapter two defines the place of conceptual design within the traditional model of the design process.

Chapter three surveys the existing automated aids for design.

Chapter four describes the scheme for representing and modelling instrument systems to be used in the knowledge based system and surveys the automated tools available for modelling in order to assess their applicability to the scheme.

Chapter five describes the organisation of the knowledge based system.

Chapter six describes the support facilities offered by the system.

Lastly, the conclusions and possible future directions for this research are discussed in chapter seven.

# CHAPTER 2

# THE ROLE OF CONCEPTUAL DESIGN

## 2.1 Introduction

This chapter defines the traditional model of the design process and the stages involved in design in order to establish the place of conceptual design.

## 2.2 The Traditional Model of the Design Process

Design is the creative process of transforming a loosely defined wish or need into a product to meet that need. A general model of the design process has been well documented (Jones, 1980; Finkelstein and Finkelstein, 1983; Pahl and Beitz, 1988; Burton, 1990). The model is based on the sequence of stages in the process which result in the transformation of the need to a suitable solution configuration. Each design stage produces solutions at a certain level of abstraction which act as the input for the next stage in the sequence. As the process proceeds the problem is defined in increasing detail and the constraints - both 'hard' and 'soft', i.e. those which must be met and those needing only to be partially satisfied - are gradually determined. The early stages of design offer the most scope for creativity as the constraints are more relaxed there, but the decisions made during the early stages are the most potentially costly ones if they are subsequently found to be wrong. Any stage of design can be modelled by the iterative procedure in Fig 2-1. This shows the sequence of design tasks which need to be performed before a solution at a certain level of abstraction is produced; the type of information flowing between the design tasks depends on

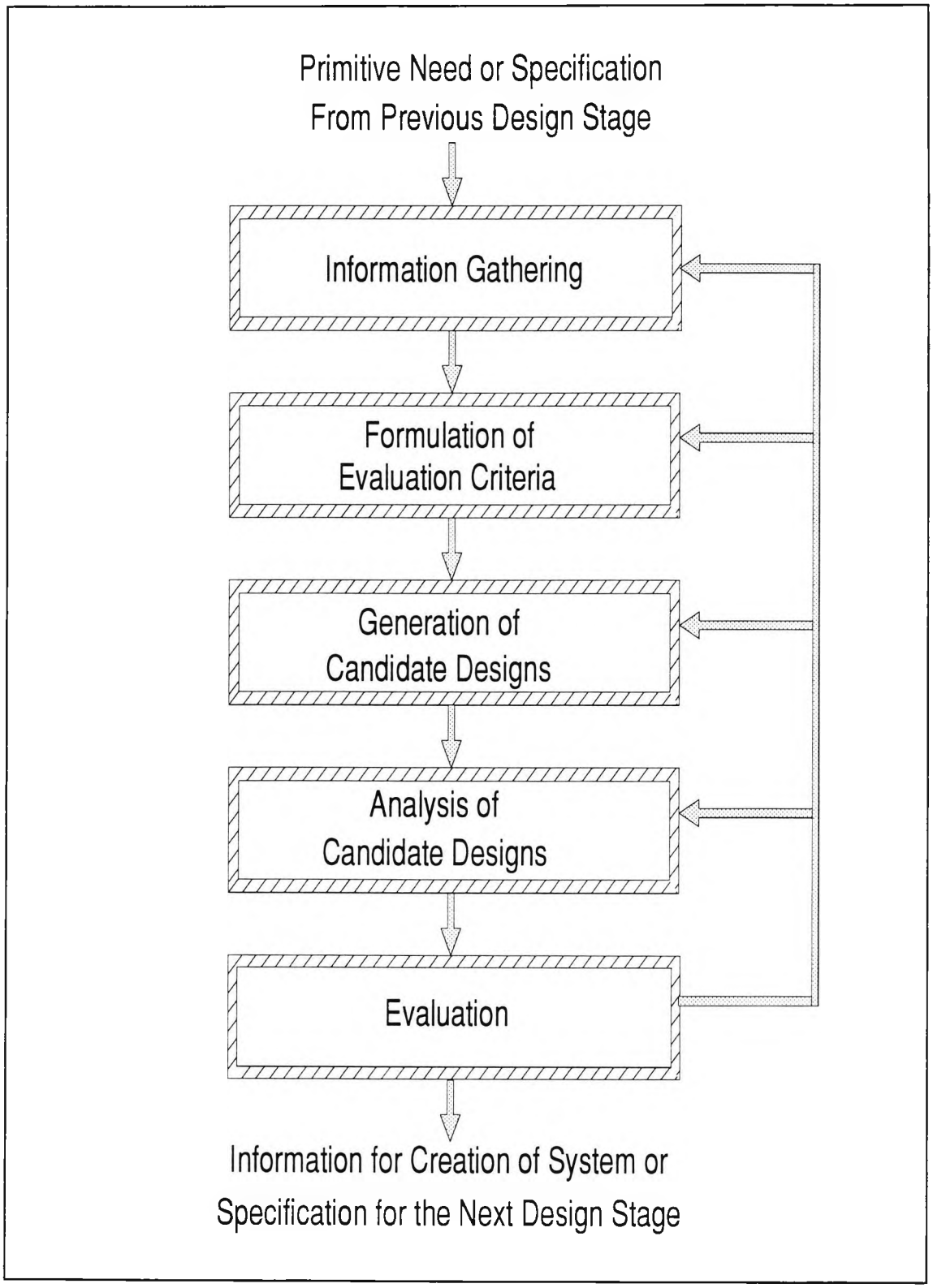



Fig 2-1 A Model for One Stage of the Design Process (Finkelstein and Finkelstein, 1983)

the level of abstraction. Any task in Fig 2-1 may be repeated if necessary, in which case the sequence of tasks following it must also be repeated before a solution is produced. It may also be decided that the initial specification of the problem at the current level of abstraction needs refining, in which case the problem is reformulated at a higher level of abstraction. The sequence of tasks in Fig 2-1 is as follows:

- Information is gathered about the problem, e.g. about the constraints, possible solutions or known similar solutions.

- The essential problems and hence the criteria for evaluating proposed designs are defined.

- A set of solutions is proposed by the designer, who is usually guided by a design methodology which should assist him by providing a framework for a systematic approach to the problem.

- The proposed solutions are analysed.

- Using the evaluation criteria one or more solutions are selected for further development at the next level of abstraction or for the creation of the system to be produced.

The above model of the design process emphasises the sequential aspects of design, but it can also be viewed from a parallel perspective. In practice a design problem is decomposed into several smaller, more manageable problems which may be carried out in parallel. Each problem, including the overall problem, can be modelled by an instance of the design process model. Also, sometimes a design stage may be started before the preceding design stages have finished, e.g. for a feasibility study. In these cases the design process is modelled by an instance of the procedure in Fig 2-1 for each active design stage.

## 2.3 The Stages Involved in Design

Fig 2-2 shows the complete sequence of activities involved in the design process. The following design stages are identified:

(i) *CLARIFICATION OF THE TASK*

The process starts with collecting information about the task requirements. Once these are understood a specification of the requirements is produced. This is a legal binding document which the final product will have to be checked against; it consists of functional and non-functional parts. The functional specification lists the requirements for the functions or tasks the system is to perform and the connections between the functions. The non-functional specification details all other requirements, e.g. those of weight, cost, size and environment.

(ii) *CONCEPTUAL DESIGN*

Conceptual design (French, 1985) is the stage of decomposing the functions in the specification into a structure of functions and identifying combinations of interconnected components or hardware subsystems which can realise the functions in the structure and fulfil the specification. The sequence of tasks in conceptual design is as follows:

- The system functions and the essential constraints on them are clarified by abstraction, i.e. the removal of unnecessary detail from the specification.

- The functions are decomposed into structures of realisable subfunctions. Fig 2-3 shows an example of the development of the functional decomposition of a fuel gauge.

- Candidate solutions to each subfunction are proposed.

- Combinations of the solutions to each subfunction are proposed.

- Suitable solution combinations are selected.

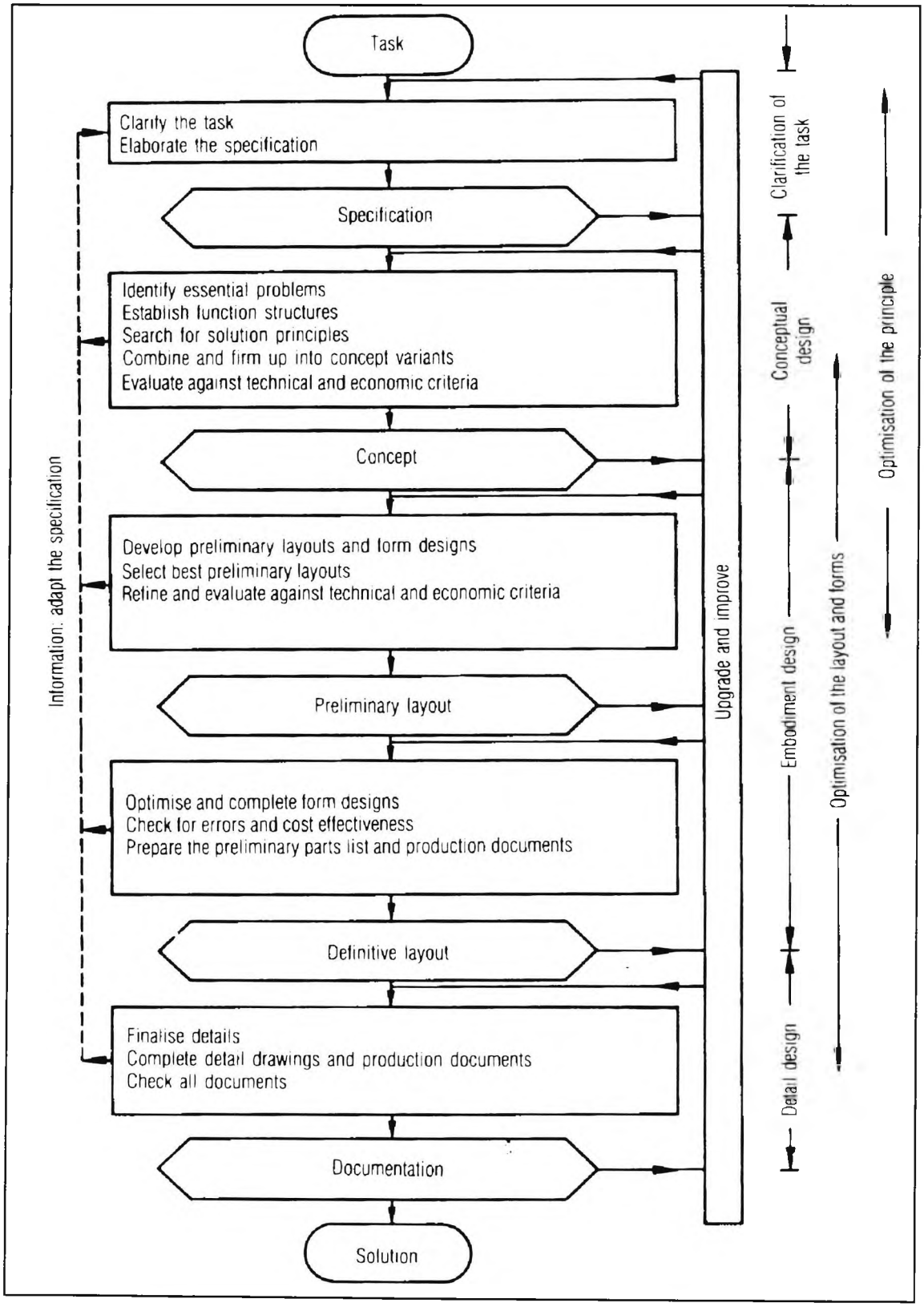



Fig 2-2 The Flow of Activity During Design (Pahl and Beitz, 1988)

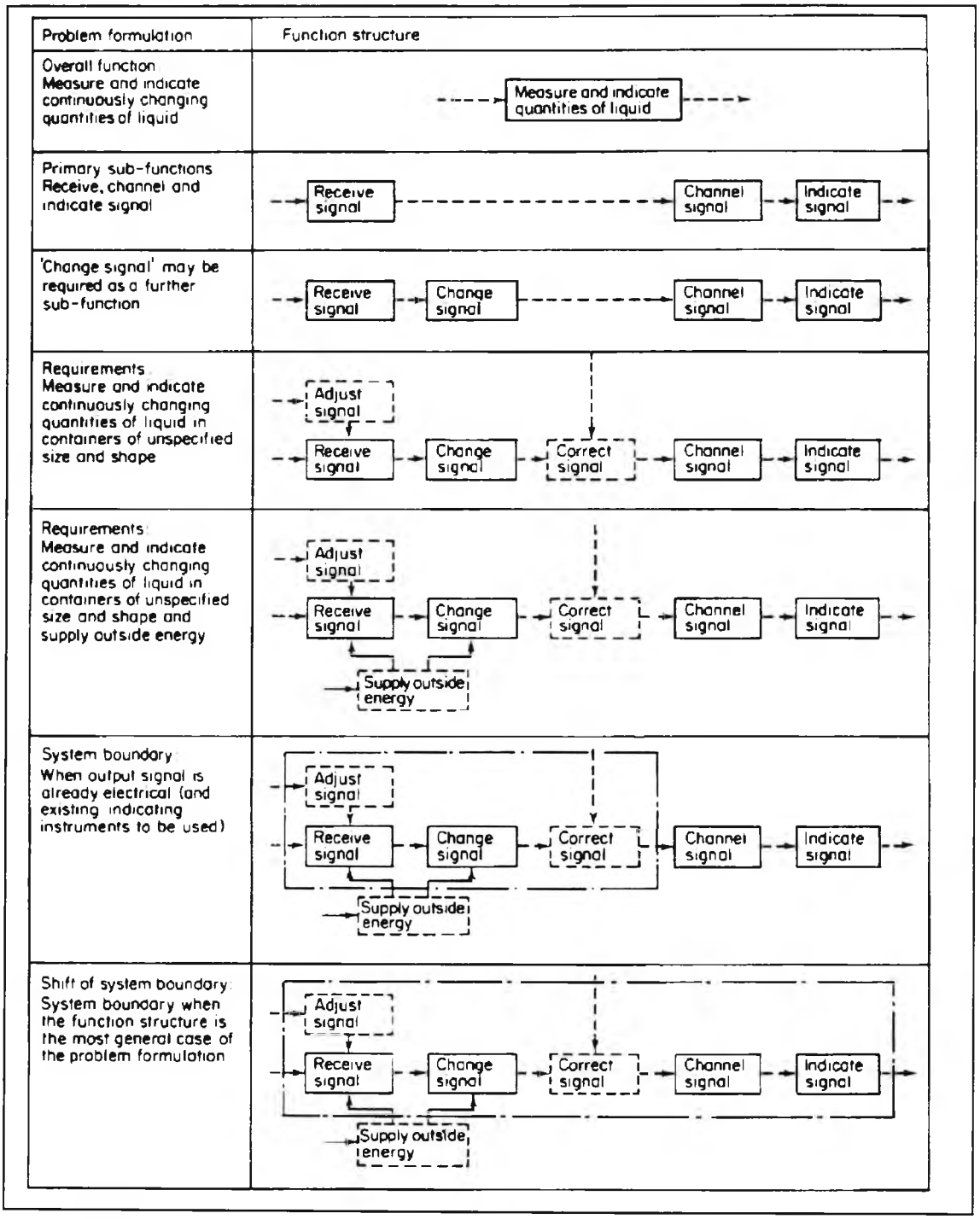



Fig 2-3 An Example of the Development of the Functional Structure of a Fuel Gauge (Cross, 1989)

- The selected solution combinations are 'firmed up' as described by Pahl and Beitz(1988), i.e. their functional and non-functional properties are analysed.

- The selected solution combinations are evaluated with respect to the technical and economic criteria.

The correspondence of these tasks to those in Fig 2-1 is shown in Fig 2-4.

(iii) *EMBODIMENT DESIGN*

Embodiment design is the phase in which the geometrical form and materials are determined for conceptual design solutions.

(iv) *DETAILED DESIGN*

In this phase the arrangement, form and dimensions of each individual part in the system is determined and all necessary drawings and documentation are produced.

## 2.4 Summary

A definition has been given of the traditional model of the design process and the specific stages involved, including conceptual design, which has been examined in more detail. The next chapter surveys automated aids for design.

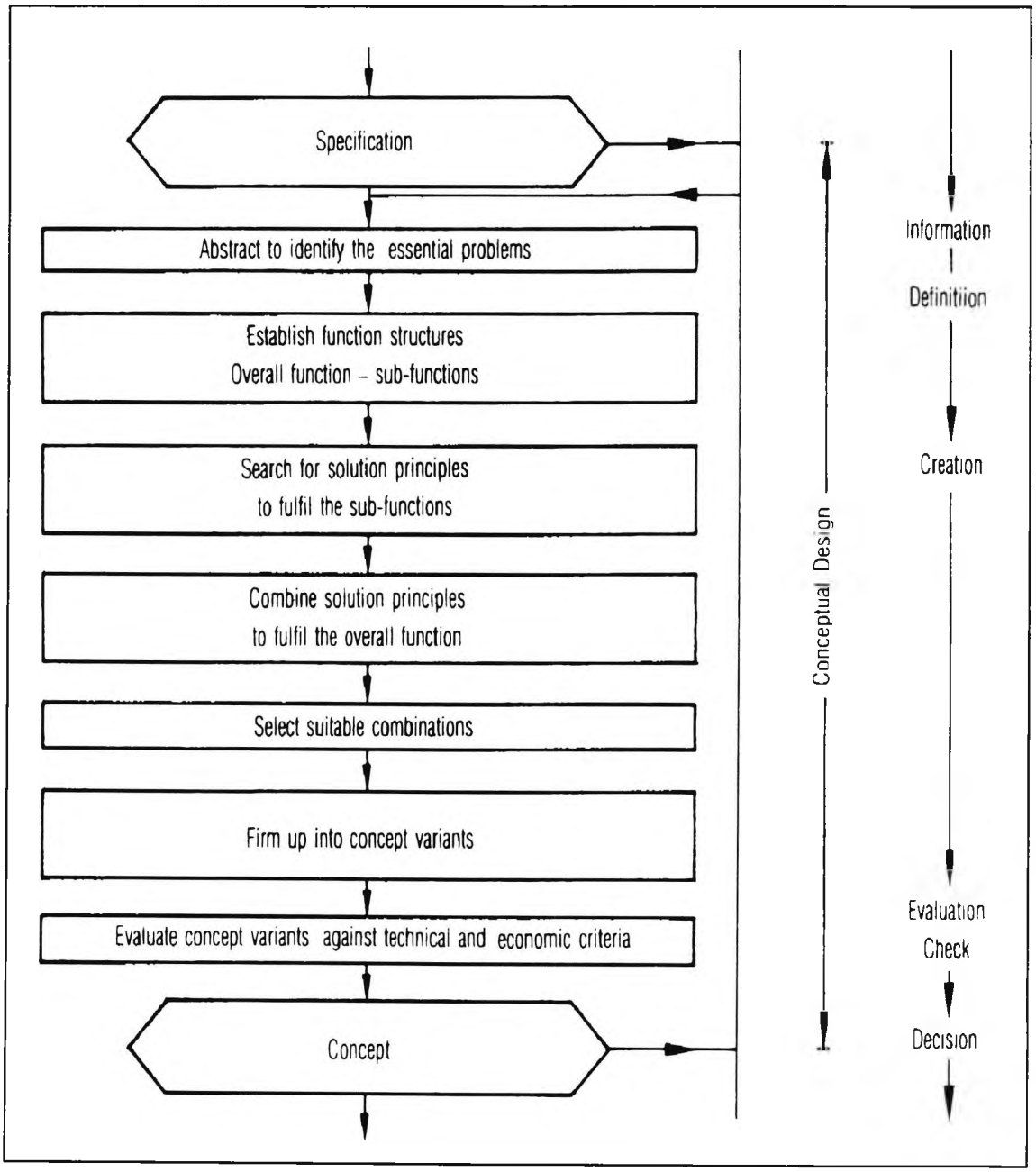



Fig 2-4 A Model for Conceptual Design (Pahl and Beitz, 1988)

# CHAPTER 3

# A SURVEY OF COMPUTER-AIDED DESIGN SYSTEMS

## 3.1 Introduction

This chapter is divided into two reviews, one of knowledge representation methods, which is carried out in order to clarify the terminology to be used in the main review, which is of automated aids for design. The latter review describes the following characteristics of design aids used in electronic, civil, mechanical and process engineering, and in metallurgy:

- The strategy employed to generate solutions
- The search procedure employed to find the knowledge stored by the system
- The method of knowledge representation used
- The method used to evaluate solutions
- How much control the designer has over the system

The purpose is to identify the characteristics of these systems which a knowledge based system for design concept generation for instrument systems

should have.

## 3.2 Methods of Knowledge Representation

The selection of a knowledge representation scheme depends on what type of knowledge is to be represented and how it is to be manipulated; sometimes a combination of schemes may be appropriate. Knowledge can be classified as declarative or procedural: declarative knowledge is factual information; procedural knowledge is knowledge of an action to be taken. It may also be incomplete or uncertain and may be represented at different levels of abstraction. Reviews of established knowledge representation techniques are found in Brachman and Levesque (1985), Frost (1986) and Jackson (1986). The following methods of knowledge representation are the main ones used.

(i) *PRODUCTION RULES*

A rule-based representation consists of a knowledge base comprised of a set of rules, each representing a chunk of knowledge, a working memory which contains facts about the current problem, and an inference engine which infers new facts from the rules. Each rule is of the form:

IF < condition > THEN < action >

The inference engine performs the following cycle of actions: it matches the conditions or actions of the rules with data in the memory, selects one of the matched rules, and executes the action part of the rule which updates the memory. There are two strategies used by the inference engine: backward chaining or forward chaining; these strategies may also be combined. A backward chaining system seeks to find the conditions necessary for an action. This involves the inference engine working backwards deriving the conditions which the initial conditions necessary for the action depend on, until the set of independent conditions which influence the action is found. In a forward chaining system the inference engine works forwards from conditions to derive as much information as possible. A rule-based representation has the advantage that a

knowledge base can be developed incrementally, but if the rule set becomes too large it is difficult to determine the effect of adding new rules; to ease this problem rules can be organised into subsets, each capable of solving a certain problem.

(ii) *STRUCTURED OBJECTS*

Jackson (1986) uses the term 'structured object' to describe "any representational scheme whose fundamental building blocks are analogical to the nodes and arcs of graph theory or the slots and fillers of record structures". It is often used to represent a classification which is stored as a tree structure and the nodes often represent objects or classes of objects which are linked by the relations *IS-A* or *A-KIND-OF*: the *IS-A* relation indicates that an object is a member of a certain class of objects; the *A-KIND-OF* relation, sometimes abbreviated to *AKO*, indicates that one class of objects is a subclass of another. In a structured object which is hierarchical information related to a node at one level of abstraction can be inferred or inherited from the information stored in the related nodes at higher levels of abstraction, e.g. it can be inferred from the hierarchy in Fig 3-1 that the shape of WEDGE18 is TRIANGULAR and the shape of BRICK12 is RECTANGULAR. The information does not have to be stored explicitly as it is inferred. If there are exceptions, i.e. cases where information should not be inherited, then this knowledge has to be stored explicitly.

Jackson (1986) describes three types of structured objects: semantic networks, frame-based systems and object-oriented networks. A semantic network (Quillian, 1968) is a structured object consisting of nodes representing concepts, which for Quillian were english words, and the nodes are connected by links which represent the relations between concepts. By searching through the network it is possible to find the relation between any two concepts, if one exists. In a frame-based system (Minsky, 1975) each node is known as a frame and consists of a number of 'slots', one of which contains the name of the object the frame represents and the rest contain the values of the attributes associated with the frame, e.g. the *'INPUT'* slot of a frame named *'BEAM'* could contain the value

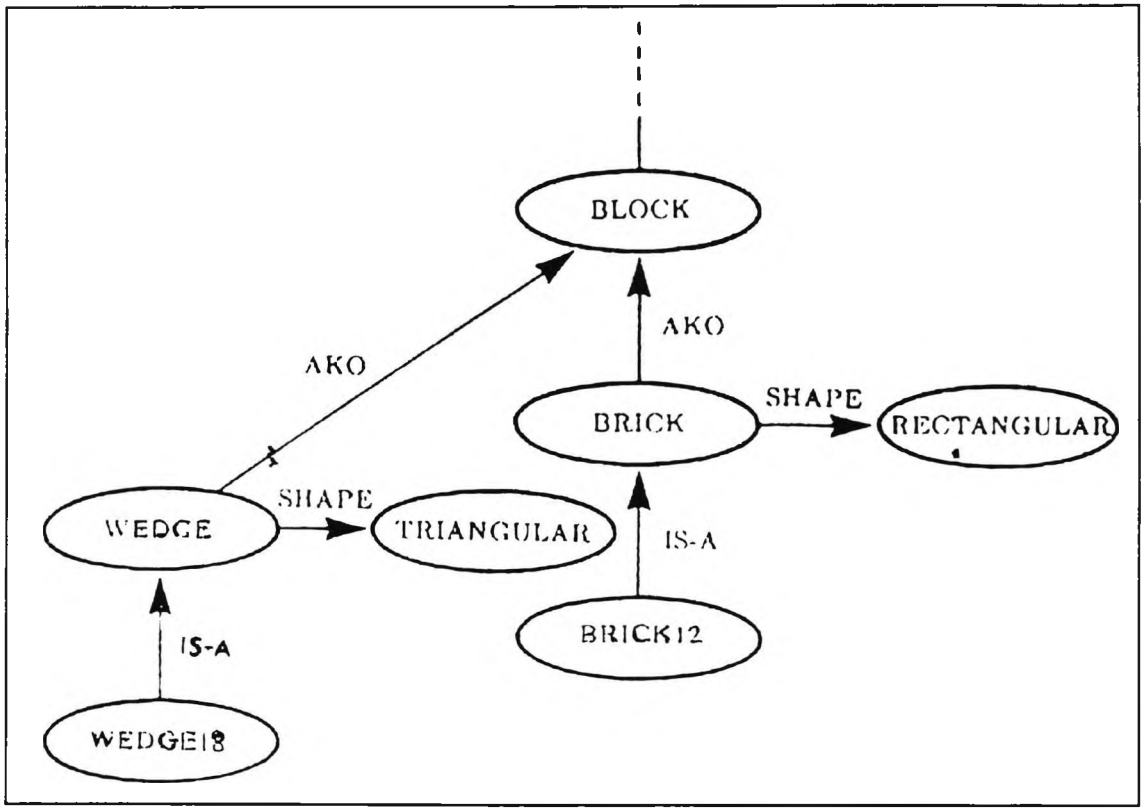



Fig 3-1 An Example of a Hierarchical Structured Network (Winston, 1984)

*'FORCE'*. The information in a slot is usually static but it can also be calculated via a procedure which is activated whenever the slot is accessed, e.g. as the volume of an object can be calculated from the height and cross-sectional area of the object a slot storing the volume of an object may activate a procedure to do the calculation when the slot is accessed; another example is when a slot containing information on the distance travelled by an object is accessed a procedure could be activated to find this out if it depends on other variables such as time. In an object-oriented network (Cox, 1986) each node is known as an object and is associated with a number of 'methods', each of which specifies a processing action to be taken when a specific message is received by the object. Tesky (1991) describes a frame-based system as a special case of an object-oriented network in which there is one main type of object, the frame, which processes two types of messages, 'display slot' and 'update slot'.

(iii) *LOGIC*

Hunter (1991) gives a survey of the many types of logic representation. A logic-based representation of a knowledge base consists of statements of fact about a certain domain, described in a formal language such as predicate calculus, and constructed using the rules for constructing expressions of the language; these rules can also be used for constructing new expressions which are deductions of theorems. The main advantage of this type of representation is that the meaning associated with a logical expression is unambiguous. However, there are disadvantages: one is the combinatorial explosion which results when attempting to prove the equivalence of two expressions; also, apart from deduction, other forms of reasoning are difficult to handle with some forms of logic, e.g. reasoning by use of defaults or exceptions.

## 3.3 Automated Support for Design

### 3.3.1 Computer-Aided Electronic Circuit Design

A functional specification of a digital circuit can be transformed to a physical description by a 'silicon compiler' which Rupp (1981) gives a definition for:

> "A silicon compiler translates a behavioural description of a function into a set of geometric images which can be used to fabricate an integrated circuit that performs that function. A sophisticated silicon compiler has the potential of allowing the design of very large circuits based on the same methodology that allows modern software compilers to generate very large complex programs."

Silicon compilers perform a combination of design tasks in one step. The obvious advantage is that they are much quicker than other more standard CAD techniques, i.e. circuit design followed by manual layout and routing of the components and their connections, but there are also some disadvantages: the designer has no access to the intermediate developments, solutions are generated from a standard library which leads them to be inefficient compared with those produced by a human designer (the greater the jump in levels of abstraction the greater the resulting inefficiency), and only one solution is produced for any given specification. They are normally used for a specific

application area, e.g. signal processing (Denyer, 1986), and when a design has to be produced quickly with little concern for minimisation of area.

Synapse (Subrahmanyam, 1986) is a system which accepts a high-level Very Large Scale Integration (VLSI) circuit specification, consisting of functional and performance specifications, and maps this into representations of custom VLSI circuits. This is an area where there are multiple views of the problem, each with their own concepts and terminology. These include functional, structural and physical considerations: functional requirements are in the form of software or logical expressions; structural requirements can be described at a number of levels of abstraction including the register transfer, switch or transistor levels; physical requirements are concerned with placement and geometry. At each level of abstraction, and for all perspectives, a specification is represented by an algebraic expression composed of a small set of algebraic primitives. The system decomposes an expression representing the specification by repeatedly transforming it to an equivalent form until it is represented at the lowest level of abstraction; typically, the transformations either change the perspective or the level of abstraction or they improve performance characteristics; all transformations are formally proven to leave the functional behaviour unchanged. The transformation of expressions is top-down with the inclusion of bottom-up correction to improve the performance and structure of a solution while still satisfying the specification. As there are many possible solutions to any given specification the system reduces the number of transformations which may be applied by adopting a strategy for transforming certain classes of expression. It is also possible for the transformations to be guided by the user, who is then responsible for proving the transformations are correct, but may be aided by the inference mechanisms of the system to do this.

VERIFY (Barrow, 1984) is a PROLOG program (Clocksin and Mellish, 1984) which checks the functional correctness of a digital circuit. This is done by finding the overall functional behaviour of a structure of components in a circuit and then comparing this with the expected overall behaviour in a specification. The

functional behaviour of a circuit or any component in it is specified by two sets of equations, one of which relates the values of the output variables of the circuit or component to the values of the input variables and state variables, and one of which relates the next values of the state variables to the current values of the input variables and state variables. The problem to be solved is to prove the equivalence of the two finite state machines which represent the overall behaviour and the expected behaviour of the circuit; this is done by symbolic manipulation of expressions representing the two finite state machines. An example of a circuit which can be checked in this way is shown in Fig 3-2. The circuit is composed of three components: a multiplexer, a register and an incrementer, each with a single output variable described by the equations *'out := if ( switch, in1, in0 )'*, *'out := contents'*, and *'out := 1 + in'* respectively. The register has a state variable with the equation *'contents := in'*. The expected behaviour is described in terms of a single output variable which is mapped to the output variable of the register, and a state variable which is mapped to the state variable of the register: the state variable has the equation *'count := if ( ctrl, in, count + 1'*, i.e. if the control signal is enabled the count variable takes the value of the input variable, otherwise the count variable is incremented; the output variable has the equation *'out := count'*. In PROLOG the structure is expressed as:

```
module(counter).

port(counter, in(Acounter), input, integer).
port(counter, ctrl(Acounter), input, boole).
port(counter, out(Acounter), output, integer).

part(counter, muxA(Acounter), mux).
part(counter, regA(Acounter), reg).
part(counter, incA(Acounter), inc).

connected(counter, ctrl(Acounter), switch(muxA(Acounter))).
connected(counter, in(Acounter), in1(muxA(Acounter))).
connected(counter, out(muxA(Acounter)), in(regA(Acounter))).
connected(counter, out(regA(Acounter)), in(incA(Acounter))).
connected(counter, out(incA(Acounter)), in0(muxA(Acounter))).
connected(counter, out(regA(Acounter)), out(Acounter)).
```

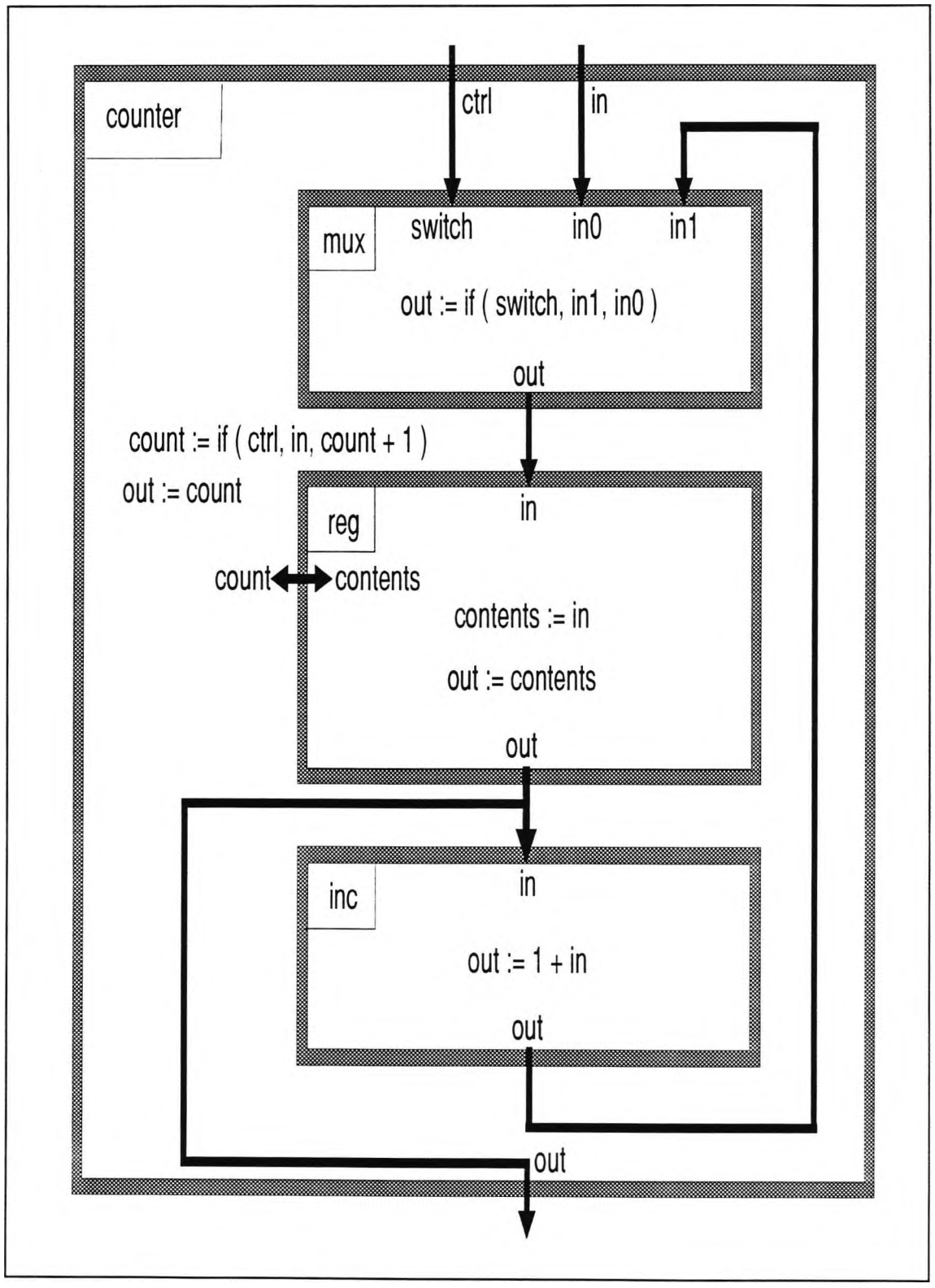



Fig 3-2 The Functional Specification of a Circuit Which can be Checked Using VERIFY (Barrow, 1984)

and the behaviour is expressed as:

```
state(counter, count(Acounter), integer).
statemap(counter, count(Acounter), contents(regA(Acounter))).

outputEqn(counter, out(Acounter) := count(Acounter)).

stateEqn(counter, count(Acounter) :=
        if (ctrl(Acounter), in(Acounter), count(Acounter) + 1)).
```

The trace resulting from proving the functional behaviour of the structure is equivalent to the expected overall functional behaviour is as follows:

```
?- verify(counter).

>>>>>> Attempting to verify counter >>>>>>
Verifying components of counter

        >>>>>> Attempting to verify inc >>>>>>
        inc is primitive (needs no verification).
        <<<<<< Success - Behaviour of inc meets its specification.

        >>>>>> Attempting to verify reg >>>>>>
        reg is primitive (needs no verification).
        <<<<<< Success - Behaviour of reg meets its specification.

        >>>>>> Attempting to verify mux >>>>>>
        mux is primitive (needs no verification).
        <<<<<< Success - Behaviour of mux meets its specification.

Verifying counter as a whole
Determining specified behaviour of counter
Specified behaviour is:

Module: counter
Inputs: ctrl(counter), in(counter)
States: count(counter)
Output equations:
        out(counter) = count(counter)
State equations:
        count(counter) = if(ctrl(counter), in(counter), count(counter) + 1)

Determining behaviour of counter from structure.
        Determining behaviour of inc as primitive
        Determining behaviour of reg as primitive
```

```
        Determining behaviour of mux as primitive
Constructed behaviour is:

Module: counter
Inputs: ctrl(counter), in(counter)
States: count(counter)
Output equations:
        out(counter) = count(counter)
State equations:
        count(counter) = if(ctrl(counter), in(counter), 1 + count(counter))

Trying to show behaviours are equivalent.
Considering equations for out(counter)
We must show that
        value(count(counter)) = value(count(counter))

Trivial identity: true

Considering equations for count(counter)
We must show that

        if (value(ctrl(counter)),
            value(in(counter)),
            value(count(counter)) + 1)
        =
        if (value(ctrl(counter)),
            value(in(counter)),
            1 + value(count(counter)))

Complexity: 25 subterms.
Trying symbolic manipulation.
Canonicalizing the identity....
Canonical form is:
        1
Identity has been established.
<<<<<< Success - Behaviour of counter meets its specification.
```

### 3.3.2 Computer-Aided Single Board Computer Design

To design a single board computer system involves working out the logic circuits required, their placement on a circuit board, and the manufacture and testing of the circuits. MICON (Birmingham and Siewiorek, 1988) is a suite of software packages which help a designer to do this. The logic circuits are generated from a design specification by a rule-based program known as M1. This guides the

designer via its own model of the single board computer design process which is based on the following steps:

1) A specification of a subsystem is given by the designer.

2) Suitable subsystems are selected.

3) The intra-subsystem construction is then determined, e.g. the structure of a memory is determined using cascadable chips.

4) The inter-subsystem integration is then performed.

5) The resulting subsystems are evaluated and the 'best' is selected as the one which minimises the following empirically developed objective function:

$$F_s = \sum_i \frac{(s_i - f_i)}{w_i}$$

where $F_s$ is the objective function, $s_i$ is the value of specification feature i, which is a desired performance, physical or electrical attribute of the subsystem, $w_i$ is a weight indicating its relative importance, and $f_i$ is the value of specification feature i for a selected subsystem.

6) Step 1) is then returned to for the specification of the next subsystem until all subsystems have been specified.

The system represents knowledge in the form of templates. A template contains information about how to achieve a certain function. Templates are organised hierarchically, the lower a template is in the hierarchy, the more specific is the function it contains information on; a template at one level in the hierarchy may have several associated templates at the next level down. The hierarchy is divided into three levels of abstraction as shown in Fig 3-3. The top level represents subsystems with reference to a standard MICON bus which provides

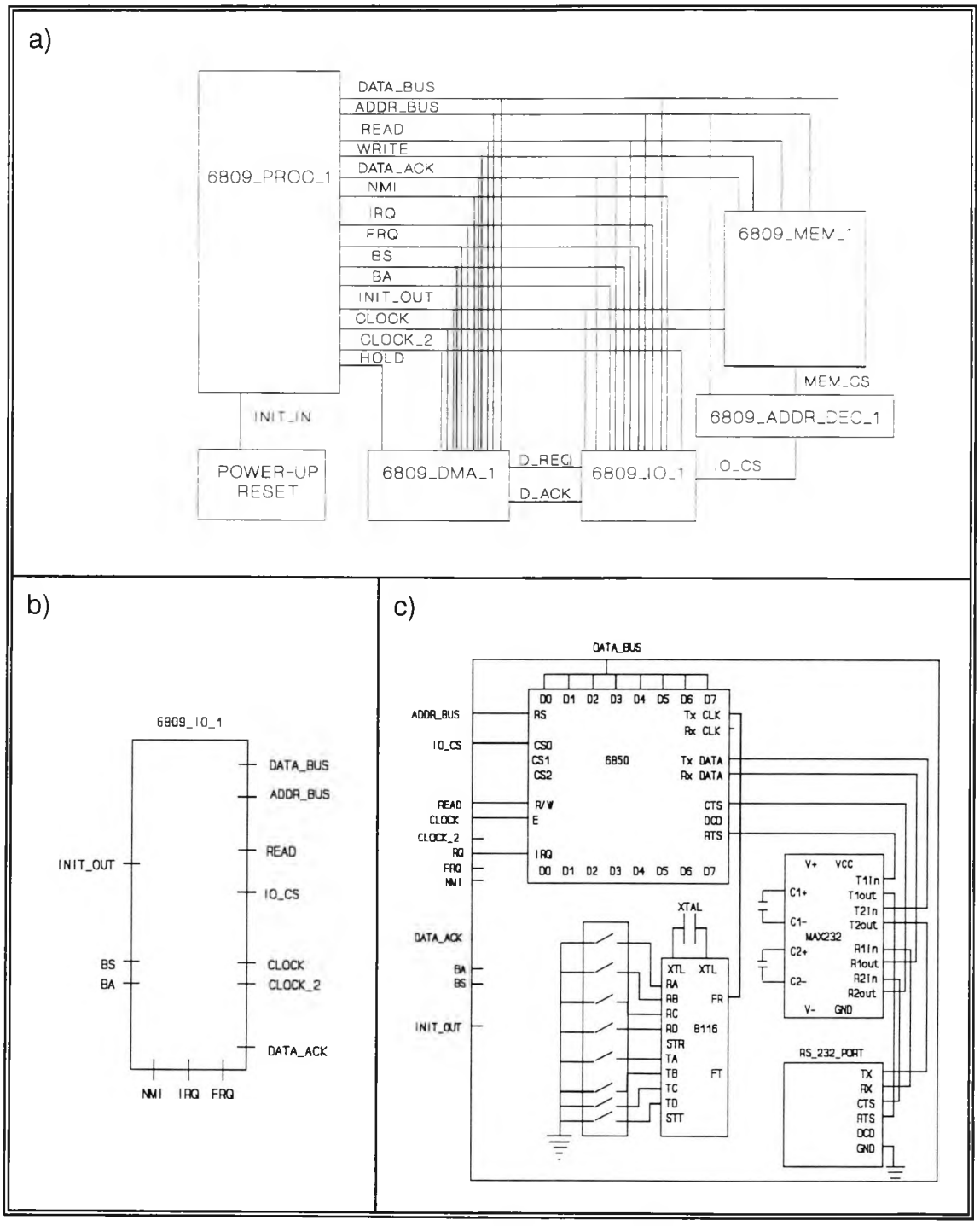



Fig 3-3 Levels of Template Representation in M1 (Birmingham and Siewiorek, 1988)

(a) With Respect to a Standard Bus Structure
(b) With Respect to a Processor Family
(c) With Respect to a Detailed Description of a Subsystem Configuration

the interconnection structure between subsystems, the next level represents subsystems with respect to a processor family, and the lowest level represents subsystems and all supporting circuitry such as resistors and capacitors. To retrieve information on how to achieve a particular function M1 starts by selecting a template at the top level and moves down the hierarchy selecting templates until a suitable template at the lowest level is found.

The software in M1 is divided into a knowledge base, which holds templates of design knowledge and a problem solver, which holds the model of the single board computer design process. Both are made up of a collection of operators, each of which performs a specific task when activated to do so. The knowledge base is formed from a collection of synthesis operators which perform the specification, subsystem selection, intra-subsystem construction, inter-subsystem design, constraint checking, and inference engine tasks. The problem solver is formed from a collection of design step operators, each of which transforms the set of variables and constraints representing one particular design state to another design state. The result of applying a design state operator is a set of goal states (usually one) which represent the current objectives of the design process; the goals are solved by the appropriate synthesis operators. During the course of development M1 can proceed along several non-conflicting lines of reasoning, e.g. once all preconditions for the memory subsystem design and the processor subsystem design are known the designs of these systems can proceed in parallel; the decision of which line of reasoning to take at any time is decided by the inference engine.

Both types of operator are implemented as a set of rules with each rule in the set describing a different situation in which the operator is to be used. The right hand side of the rules are postconditions which define the action of the operator. The left hand side of the rules are preconditions which define when the operator can be applied. Operator preconditions are satisfied either by the values in the specification given by the designer or as a result of the application of synthesis operators. Design state operator preconditions consist of the design state the

operator is to be activated from. Synthesis operator preconditions consist of the goal the operator is to be activated from and the design state in which it is to be used; a set of preconditions for the same synthesis operator are distinguished by the different design states in each precondition. There is no explicit sequencing of operators; they are applied whenever their preconditions are met, but as all preconditions must be unique, at most only one operator in any set which perform the same function will be able to be applied at any one time. The uniqueness of operator preconditions enables new design steps and synthesis knowledge to be added whenever necessary without altering the rest of the software.

### 3.3.3 Computer-Aided Preliminary Structural Design

HI-RISE (Maher, 1988a) generates alternative structural configurations which meet an architectural specification consisting of functional specifications for the lateral load resisting and gravity load resisting subsystems of a building. The specification is input to HI-RISE in the form of a three-dimensional grid as shown in Fig 3-4. This describes the spatial constraints the building must satisfy, including the number of stories and bays in each direction, the dimensions of the bays, and the location of vertical service shafts or internal spaces. Other information specified includes the intended occupancy of the building and the wind and live load. Design knowledge is stored by the system in the form of frames and heuristic rules: the frames contain generic descriptions of the design components used; the rules hold a strategy for decomposing an initial specification and information on the constraints for using components and for the integration of a system. The expertise in HI-RISE is derived from the approximation analysis techniques and design heuristics described in Lin and Stotesbury (1981).

The design of each subsystem is decomposed into the following subtasks: synthesis, analysis, parameter selection, evaluation and system selection; these subtasks are carried out sequentially, in the same manner for both subsystems. The lateral load resisting system is designed by HI-RISE first as the design of

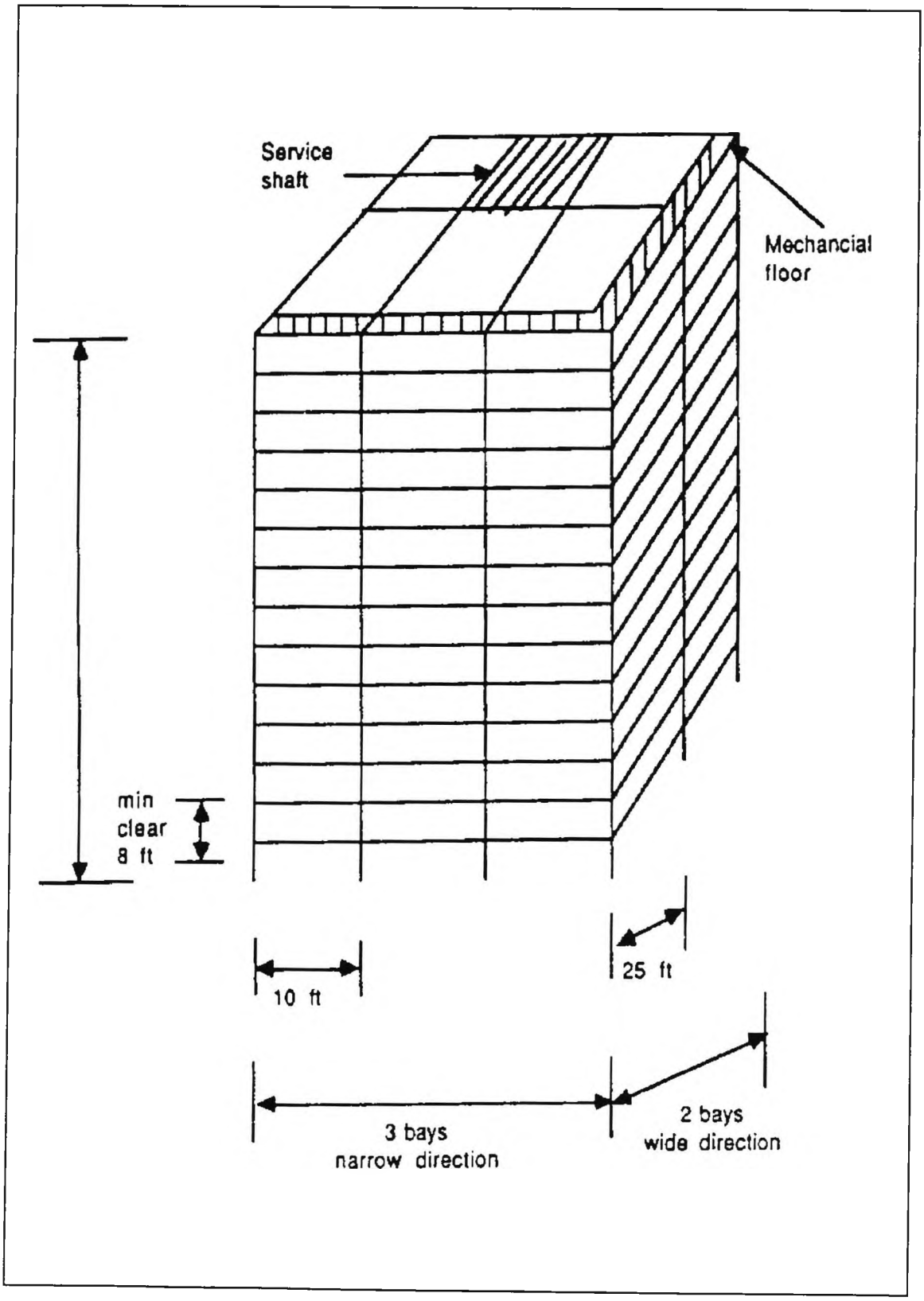



Fig 3-4 The Specification Input to HI-RISE (Maher, 1988a)

that part of the system is thought to be the most important. This enables the constraints for the gravity load resisting system to be precisely defined, but also means that design heuristics have to be used to estimate the results of the design of the gravity system which are relevant to the design of the lateral system, e.g. the type, depth, and weight of the floor system. The synthesis task generates valid combinations of generic design components. This is done by a procedure which involves a search through a hierarchical representation of the design decisions to be taken. Associated with each level in the hierarchy is a set of elimination rules which check the feasibility of different design decisions at a certain level of abstraction given the design decisions already taken and the requirements in the specification. An example of an elimination rule is: 'if the number of stories is less than forty and the two-dimensional lateral system is rigid frame then the alternative is eliminated'. A feasible decision is added to the design description and the search proceeds to the next level in the hierarchy until all levels have been considered. The synthesis task generates generic solutions for a lateral load resisting system by taking decisions about the following:

- Three-dimensional subsystems, e.g. core or orthogonal 2-D systems.

- Two-dimensional vertical subsystems, e.g. braced frame, rigid frame or shear wall.

- Materials, e.g. steel or reinforced concrete.

- Locations of the lateral load resisting subsystem (these are decided by heuristic rules).

- The associated components.

A generic solution for a gravity subsystem is generated by the synthesis task taking decisions about the following:

- Two-dimensional subsystems, e.g. reinforced concrete, steel deck, prefabricated panels, or waffle grid.

- Support types.

- Subdivide types.

- The associated components.

A completed design description is then checked for overall feasibility by the analysis stage. This performs an approximate analysis of the configuration and generates the constraints applicable for the parameters of the components. The parameter selection task then checks the initial values of the component parameters against the constraints produced by the analysis task. If the initial parameter values cannot be used heuristic recovery rules are applied to revise the values of the parameters until they are acceptable. Both subsystems are then evaluated using a function, the value of which depends on a linear weighted combination of the design features of the subsystem; the weighting factors for each feature may be determined by HI-RISE or they may be specified by the designer. When all feasible alternatives have been generated the system selection task is invoked which presents the designer with the details of each alternative and the associated cost determined by the evaluation function; the designer then selects one of these alternatives. The default selection is the alternative determined to be the 'best' by the evaluation function, but this decision can be overridden by another selection. Fig 3-5 shows a set of alternative solutions which are generated by HI-RISE in the form of a tree structure. The nodes in the tree represent design selections and the links represent relationships between the selections. There are three types of link in the tree which are *is-alt*, *part-of* and *uses* links: an *is-alt* relationship indicates the descendants of the node form alternative configurations, a *part-of* relationship indicates that the descendants of a node are all part of the same configuration, and a *uses* relationship connects a constraint with the subsystem or component

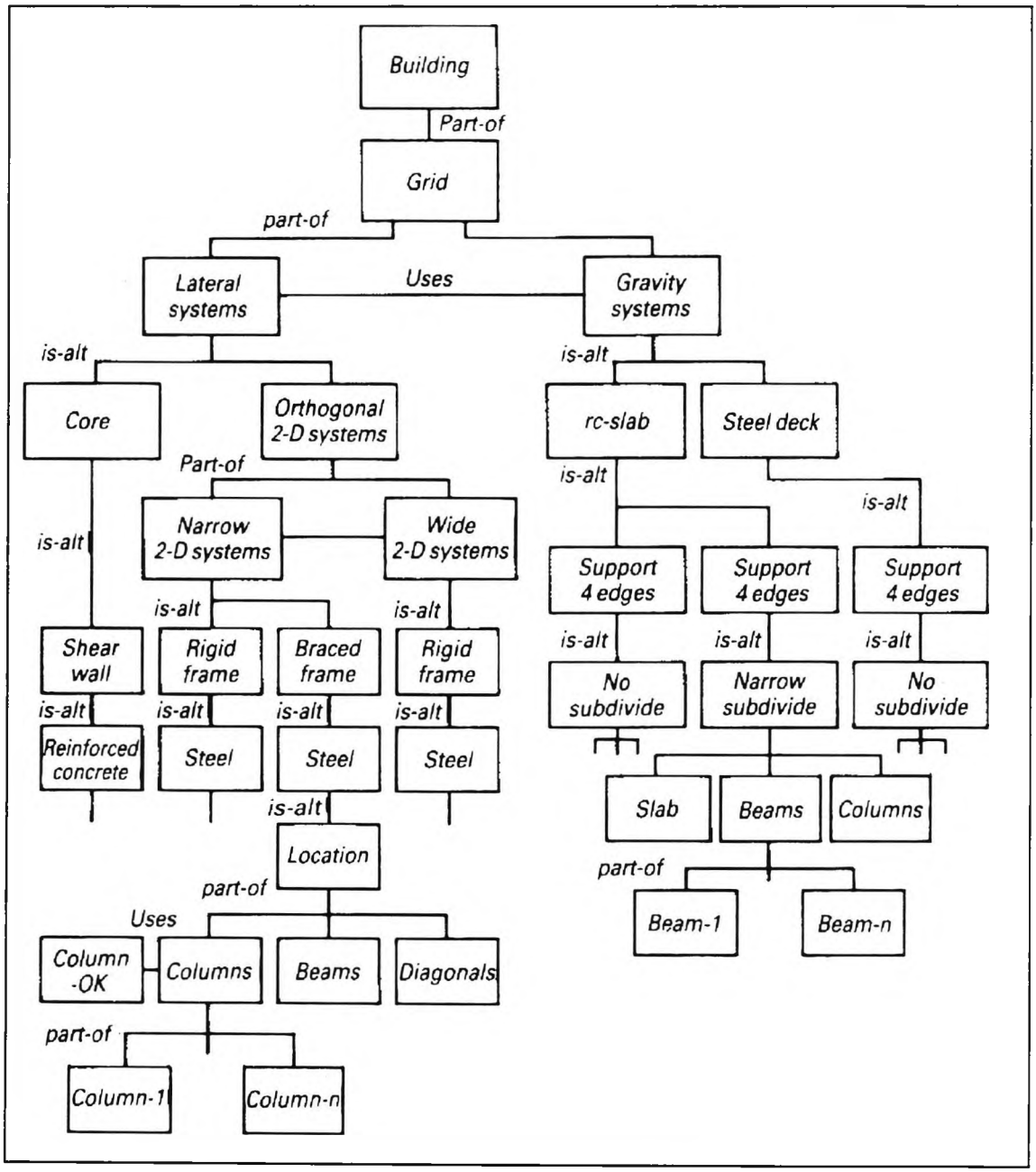



Fig 3-5 An Example of a Solution Tree Generated by HI-RISE (Maher, 1988b)

it constrains.

The research on HI-RISE has also lead to the development of EDESYN (Maher and Longinos, 1987), a domain-independent expert system shell for similar

design problems; this can generate design alternatives to meet a given specification by searching a knowledge base which has to be supplied; the knowledge base will contain a hierarchical decomposition of design descriptions and heuristic constraints for them.

### 3.3.4 Computer-Aided Aluminium Alloy Design

Aluminium alloy design involves solving several conflicting subproblems, each representing an incomplete view of the design. ALADIN (Hulthage, Farinacci, Fox and Rychener, 1988) is a knowledge based system that aids the design of aluminium alloys for aerospace applications. It represents the stored knowledge about the problem in the following spaces:

- the space of alloy properties, known as the property space

- the space of alloy microstructures, known as the structure space

- the space of alloying elements, e.g. copper or magnesium, known as the composition space

- the space of thermo-mechanical alloy manufacturing processes, known as the process space

- the meta space, which holds knowledge about the design process and control strategies for design

Partial models for alloy design are represented by sets of rules, each containing the known relations between two design spaces. The rule sets link all design spaces with the exception of the meta space (see Fig 3-6). Each rule set can propose and verify hypothesis of relations between two design spaces at different levels of abstraction and in either direction. For example, the rule set linking structure and composition can propose alloying elements which enable a specified structure or it could propose structures for specified alloying

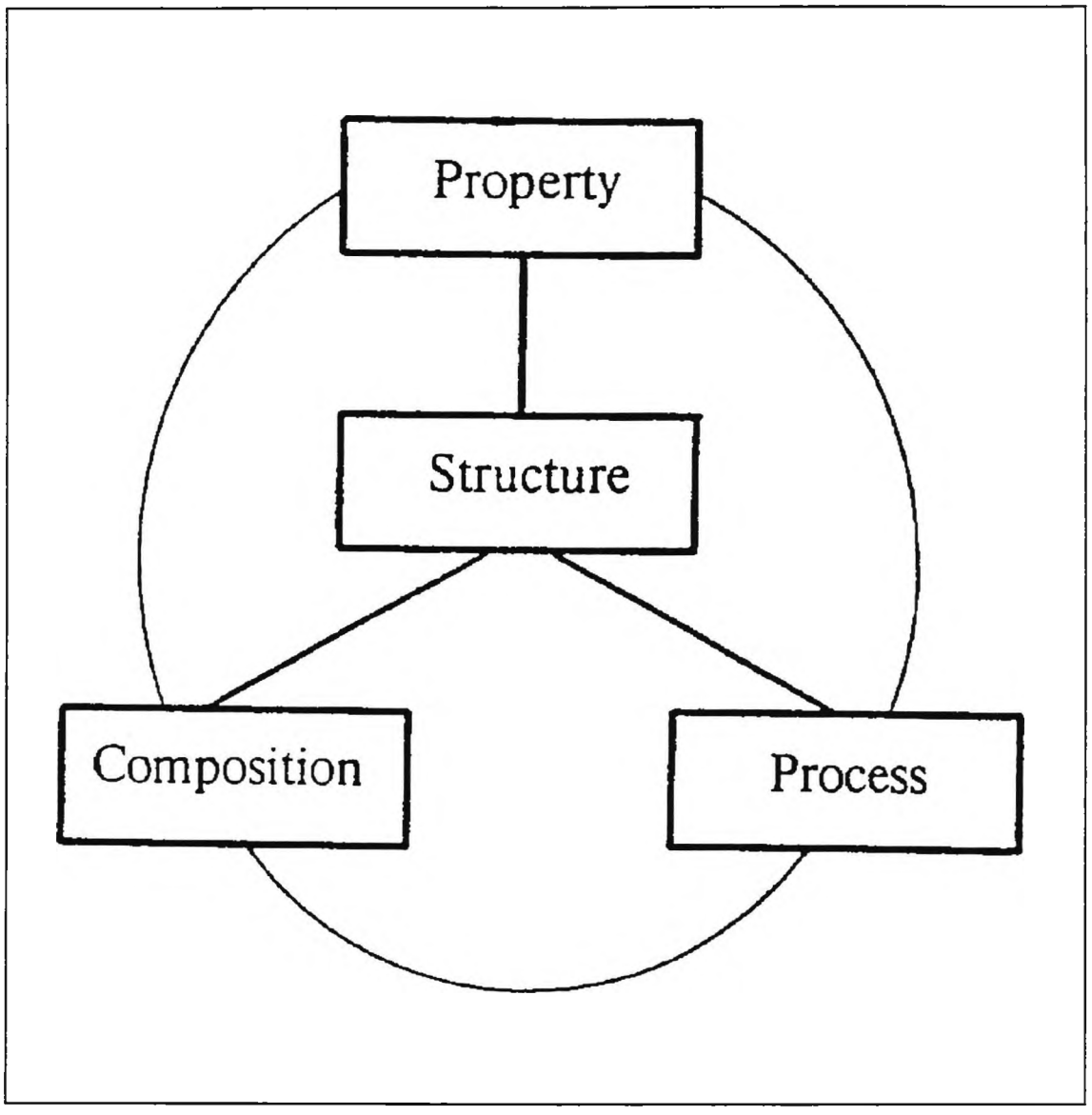



Fig 3-6 The Design Spaces Linked by Domain Knowledge in ALADIN
(Hulthage, Farinacci, Fox and Rychener, 1988)

elements.

The designer starts by specifying the desirable properties of the target alloy; these act as the initial constraints to the problem. The designer may also indicate an application area; this allows the system to select a design strategy. Plans to solve the current design problem are then proposed in the meta space by a

system which schedules the rule sets to generate hypotheses with specific ranges or values for unknown design variables. As this procedure progresses the hypotheses generated become more precise and the generation of new hypotheses is constrained. The problem is tackled across the design spaces by the following cycle of actions:

1) Evaluate the current hypothesis with respect to the necessary constraints on properties to see where it falls short of the target; the result is a set of estimates and a focus on particular properties of interest. If the current hypothesis meets the target then the problem is solved.

2) Generate hypotheses in order to meet a given target or combination of target properties; the result is a set of hypotheses with initial credibilities for selection purposes. The credibilities are estimated by more detailed evaluation; the decision as to how much more evaluation is necessary is made in the meta space.

3) Select the best hypothesis to pursue and go to step 1).

Alternatively, as with synapse (Subrahmanyam, 1986), the system may be operated manually by a metallurgist. In this mode the system leaves control to the user who guides a search in the direction he wants, each time selecting from the current options available.

### 3.3.5 Computer-Aided Preliminary Ship Design

Preliminary ship design is characterised by an iterative decision-making procedure to determine the values of the design variables, e.g. the size, speed, length and depth of a ship, which satisfy the design requirements; these are specified in terms of the required values for certain design variables. The iteration is due to assigning values to design variables, evaluating them against design objectives and modifying the design variables as a result of the evaluation. Akagi and Fujita (1987) describe an expert system which aids this

process. The system is divided into a general subsystem, which could also be used as a shell for other design problems, and a specific subsystem which, in this case is for ship design. Control of the system is done by the designer specifying in advance a sequence of design steps which state, for example, when design variables are to be determined and, when and which results are to be displayed. Evaluation of design variables is carried out by a set of diagnostic rules which advise the user during the design process. An example rule might be: if the calculated ship weight is 1.05 times greater than that of the required value then the calculated value is diagnosed 'too large'. As determining the values of design variables is very much a trial and error process software to support the decision-making is built into the system. It is possible to perform a sensitivity analysis to determine the effects of a unit change of one variable on any other related variables. Fig 3-7(a) shows the influence of a unit change in the variable 'length' on the related variables. Another option available is the optimisation of design variables with respect to an objective function by sequential linear programming as shown in Fig 3-7(b). In this example the value of 'breadth' satisfying the required value of 'approx deadweight of 38000 tons' is found to be 27.26 m.

Design variables and the relationships between them are modelled by an object-oriented network. Each design variable is represented by an object which can receive messages from and send messages to other objects in the network. A message represents a request for a function associated with the message to be performed by the object receiving the message; the object invokes the appropriate algorithm which may in turn require messages to be sent requesting other information, e.g. when the calculation of a design variable is required a message is sent to the corresponding object requesting this to be done and if the values of other variables are needed in the calculation messages are passed to the corresponding objects. Message-passing is also used to alter the values of design variables and to delete the values in related design variables. There are also other objects in the network apart from those representing design variables: some hold ship data; some supply information on the graphic representation of

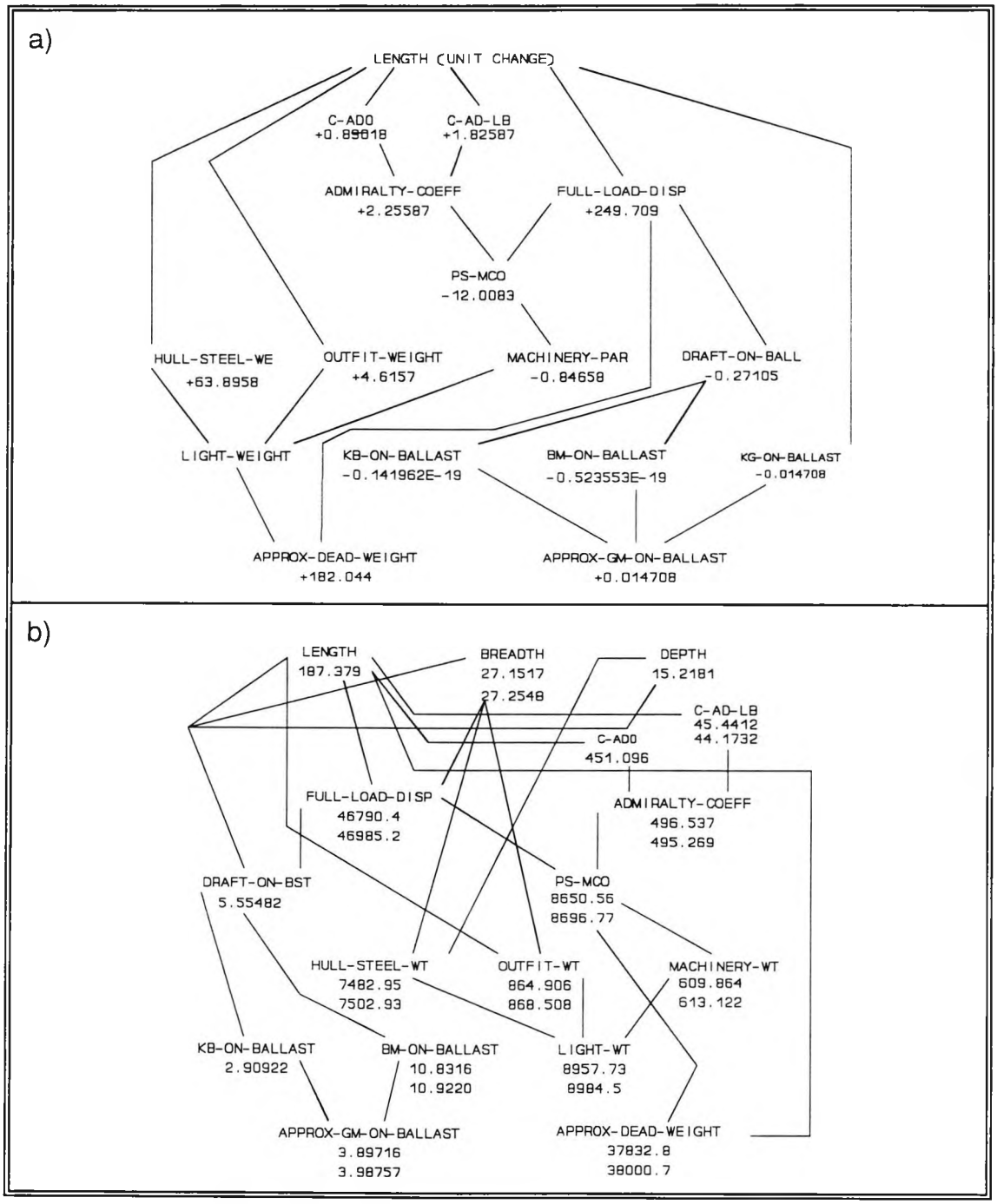



Fig 3-7 Support for Preliminary Ship Design (Akagi and Fujita, 1987)

(a) The Effects of a Unit Change in 'length' on Related Ship Design Variables

(b) The Effects of Optimisation Using Sequential Linear Programming

a ship or other information involving complex computations; these kinds of calculations are carried out via an interface to FORTRAN procedures.

### 3.3.6 Computer-Aided Process Plant Design

Lukas and Dixon (1993) describe a knowledge based engineering system for process plant design. The system employs an object-oriented network which stores knowledge on all aspects of the design of industrial plants and control systems for them including functional, component, physical, i.e. geometric representation, and performance knowledge. The objects in the network contain information on physical or abstract entities used in process plant design, e.g. an object could contain information on the capacity, pressure and motor size of a pump. The information held in an object may be in the form of numbers, references to other objects, or can be computed as a function of other properties of the same or other objects; it may also be organised hierarchically, e.g. when an object representing a centrifugal pump inherits the properties of an object representing a pump. Fig 3-8(a) shows an object-oriented description of a distillation column; the blocks represent objects which form part of the distillation column at a certain level of abstraction, the heavy lines represent part-subpart relationships, and the lighter lines represent relationships between properties of different objects.

Process plant design is divided into three stages: functional design, detailed design and equipment layout. The output of the functional design stage is a process flow diagram of a plant which the designer creates via an interface. This consists of symbols representing specific unit operations such as heat exchangers and tanks, and lines representing connecting streams (see Fig 3-8(b)). Each item in the process flow diagram is represented by an object which can determine its properties based on the objects it is connected to and the objects which are connected to it. The objects also contain algorithms for estimating their cost, and the information required to drive a process simulator to simulate the steady state behaviour of the plant; they also contain design rules to validate a design at the process flow diagram level. Following the entry and

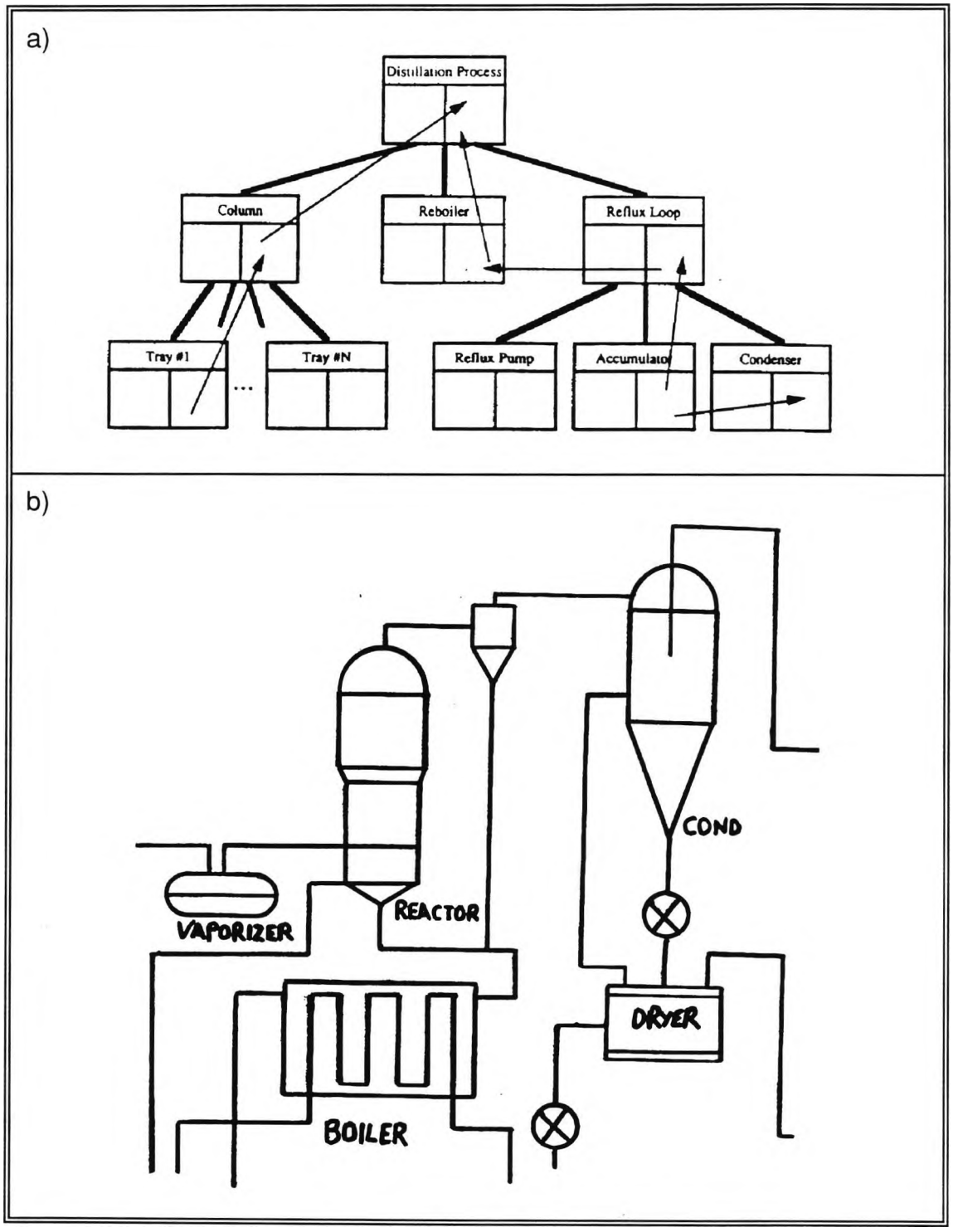



Fig 3-8 Support for Process Plant Design
(Lukas and Dixon, 1993)

(a) An Object-Oriented Representation of a Distillation Column

(b) Part of a Process Flow Diagram Model Definition

validation of the process flow diagram the system carries out the detailed design. Each object in the process flow diagram expands into objects representing all subparts of the process flow diagram and the new objects determine their properties; then, if possible, each object representing a subpart is expanded into objects representing more detailed components, their properties are determined, and this process continues until all detailed design components and their associated properties are determined. Control logic and measurement points are also generated automatically although an interface is provided for the designer to fill in extra details. The remaining task is to choose the position of the equipment and the connecting piping in the plant site and to document the resulting physical layout; automatic and semi-automatic facilities are also provided for this and the resulting solution is validated by a computer-aided drafting package which applies previously existing test routines to the solution.

## 3.4 Conclusions

Automated aids for a variety of design applications have been surveyed and reviewed. Comparing the techniques they employ a number of desirable features of an instrument system design concept generation aid emerges:

- The system should be easy to use.

- A designer should be able to use the system as an aid while applying their own approaches to solving a problem. They should be in full control of the system and not constantly prompted for information until a solution is produced.

- The system should be able to efficiently search for possible solutions and verify the functional correctness of a proposed solution.

- Empirical design rules should not be built in to the system; these may not always be applicable and could also be different for different designers.

- Evaluation of proposed solutions should be done by the designer who may

request help to do this from the system.

A functional modelling scheme for instrument systems is described in the next chapter.

# CHAPTER 4

# FUNCTIONAL MODELLING OF INSTRUMENT SYSTEMS

## 4.1 Introduction

This chapter describes a scheme for the functional modelling of instrument systems which is to be used in the knowledge based system to be described in the next two chapters. The scheme models the energy flow in the functional decomposition of an instrument system. The chapter starts with a definition of an instrument system and then describes classifications of the energy flow and functions to be modelled. This is followed by a description of how the signal flow in a configuration of functions can be modelled, a survey of automated tools for modelling, and then an assessment of their applicability for implementing the scheme, which is then described.

## 4.2 A Definition of an Instrument System

An instrument system (Abdullah, Finkelstein, Khan and Hill, 1993) is a system which acquires, processes and effectuates information in the physical world. The sequence of operations performed by an instrument system is shown in Fig 4-1 and is as follows:

1) Information is acquired by measuring the attributes of objects or events in the world from their physical input signals; this is done by a sensing element or transducer which senses the inputs (known as measurands) and converts them to another energy form.

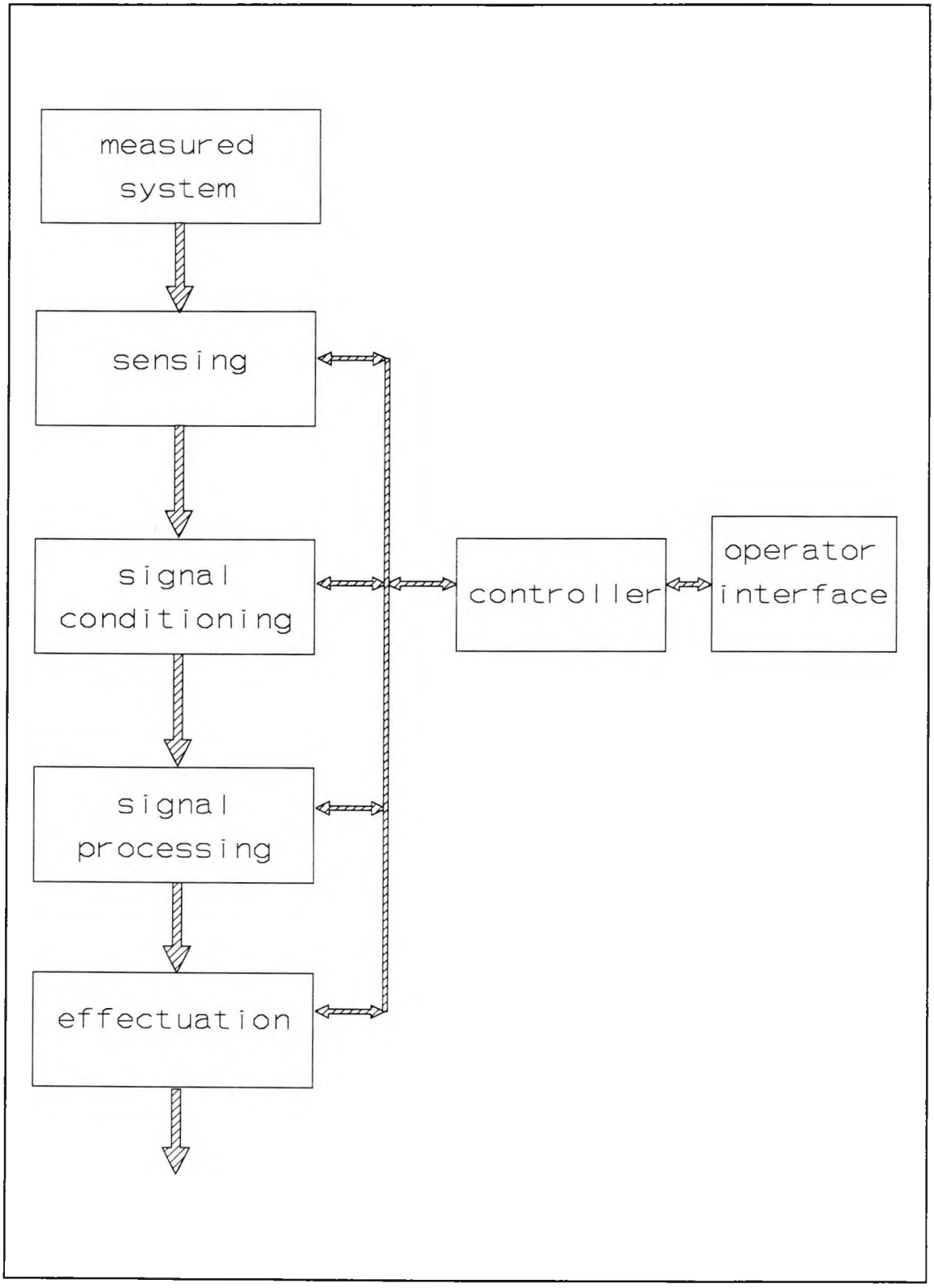



Fig 4-1 The General Form of an Instrument System (Abdullah, Finkelstein, Khan and Hill, 1993)

2) A signal conditioning element then converts the sensed outputs to a form suitable for further processing by, for example, amplification, filtering or scaling.

3) A signal processing element then finds a mapping from the measurements taken to physical output signals which describe the measurements.

4) Finally, an effectuation element gives effect to the system mapping by converting the signal processing output to physical output signals.

The overall energy transformation can be performed by a configuration of components, each of which fulfils an instrument system function and is associated with one or more of the above operations. The system is controlled via a man-machine interface connected to a controller which adjusts the operations in the way requested.

## 4.3 Representation of Energy Flow in Instrument Systems

The energy flow through an instrument system or any part of it may be classified according to the type of model which is best suited to describing the functional behaviour of the system, i.e. the relationship between the input and output energies. In all cases the energy flow can be represented in terms of the energy flowing in and out of the 'ports' attached to a one or more elements which represent the system to be modelled. The following system models are possible:

(i) *LUMPED PARAMETER SYSTEMS*

A lumped parameter system representation is appropriate when the time for the energy to traverse an object is short compared with the period associated with the highest frequency of input energy. The output energy is represented as a single time-varying quantity, an example of which is the displacement of a spring when a force is applied to it. This behaviour is described by an ordinary differential equation.

(ii) *DISTRIBUTED PARAMETER SYSTEMS*

A distributed parameter system representation is used when the time for the energy to traverse an object is of the same order as the period associated with the highest frequency of input energy. In this case the output energy varies over the dimensions of the system. An appropriate example is the temperature variation in a long metal rod heated at one end. This type of behaviour is described by a partial differential equation and could be approximated by several lumped parameter system models, each approximating the behaviour over a small length or area, depending on the dimensions of the system. The output energy is then represented by several time-varying quantities, each of which describes the variation of energy for a certain length or area. Alternatively, the output energy could be represented as a single time-varying quantity which describes the average energy density.

(iii) *RAY APPROXIMATION SYSTEMS*

A ray approximation system description is used when the time for the energy to traverse an object is long compared with the period associated with the highest frequency of input energy. In this case the output energy travels in approximately straight lines and varies with direction. An appropriate example is the energy radiated from a light source. This type of behaviour is described by a partial differential equation and could be approximated by several lumped parameter system models, each approximating the behaviour in a small region. The output energy is then represented by several time-varying quantities, each of which describes the variation of energy for a certain region. Alternatively, as with a distributed parameter system representation, the output energy could be represented as a single time-varying quantity which describes the average energy density.

The energy flow through a lumped parameter, distributed parameter, or ray approximation system can be represented by any one of the following functional models of energy flow which have been identified by Finkelstein and Watts (1978):

(i) *POWER FLOW MODELS*

These relate the power flowing in and out of the ports of an element.

(ii) *SIGNAL FLOW MODELS*

Information is transmitted via the signal present in a power flow. Signal flow models relate the signals flowing in and out of the ports of an element.

(iii) *INFORMATION FLOW MODELS*

The information in a signal can be described as a sequence of symbols, each symbol representing a piece of information at a certain level of abstraction, e.g. a television signal may be considered as a sequence of pixel values, lines or frames. Information flow models represent the information flowing in terms of symbol flow.

Each model considers the relationship between the input and output at a certain level of abstraction and can be verified by experiment. This thesis deals with functional modelling of instrument systems at the signal flow level of abstraction.

### 4.3.1 Energy Flow in Lumped Parameter Systems

Energy flow in a lumped parameter system can be described in terms of a pair of associated variables whose product is power and one of which carries a signal; it should be noted that thermal systems are usually modelled in terms of the pseudo variables heat flow rate and temperature although their product is not power. These variables are termed energy rate variables and are denoted $\dot{x}$ and $\dot{y}$ in their generalised form. The $\dot{y}$ variables act at a single point and can be said to travel 'through' an element and so are termed through variables. The $\dot{x}$ variables act between two points and can be said to act 'across' them and so are termed across variables. Table 4-1 shows a classification of various forms of energy into through and across variables. Each energy rate variable has a corresponding energy state variable which is denoted y for through variables and x for across variables; these describe the energy stored in an element. Energy rate and energy state variables are related by the following equations:

| | Through Variables | | Across Variables | |
|---|---|---|---|---|
| Class of System | Energy Rate Variable $\dot{y}$ | Energy State Variable $y$ | Energy Rate Variable $\dot{x}$ | Energy State Variable $x$ |
| Mechanical Translational | Force f [N] | Momentum h [kg m $s^{-1}$] | Velocity $\dot{x}$ [m $s^{-1}$] | Displacement x [m] |
| Mechanical Rotational | Torque T [N m] | Angular Momentum H [kg $m^2$ $s^{-1}$] | Angular Velocity $\dot{\phi}$ [rad $s^{-1}$] | Angular Displacement $\phi$ [rad] |
| Electrical | Current i [A] | Charge q [C] | Voltage v [V] | Flux Linkage $\lambda$ [Wb turn] |
| Fluid Flow | Volume Flow Rate $\dot{g}$ [$m^3$ $s^{-1}$] | Volume g [$m^3$] | Pressure p [N $m^{-2}$] | Pressure Momentum P [kg $m^3$ $s^{-2}$] |
| Thermal | Entropy Flow Rate $\dot{S}$ [kJ $K^{-1}$ $s^{-1}$] | Entropy S [kJ $K^{-1}$] | Temper-ature $\theta$ [K] | – |
| Thermal (Pseudo) | Heat Q [J] | Heat Flow Rate $\dot{Q}$ [J $s^{-1}$] | Temper-ature $\theta$ [K] | – |

Table 4-1 Classification of Through and Across Variables in Physical Systems (Finkelstein and Watts, 1983)

$$rate\ variable = \frac{d}{dt}(state\ variable)$$

$$change\ in\ state\ variable = \int_{t1}^{t2}(rate\ variable)\,dt$$

In an interconnected network of lumped elements the through and across variables at each port obey the following generalisations of Kirchhoff's laws:

(i) Kirchhoff's generalised vertex law (expresses continuity of mass and energy):

*The algebraic sum of the through rate variables acting at any point in a physical system is zero.*

(ii) Kirchhoff's generalised circuit law (expresses continuity of space):

*The algebraic sum of across rate variables around any closed circuit of a physical circuit is zero.*

Energy flow in lumped parameter systems could also be described in terms of 'effort' and 'flow' variables. For systems other than mechanical ones an effort variable is the same as an across variable and a flow variable is the same as a through variable; in mechanical systems the opposite is true. For all systems the ratio of an effort variable to the corresponding flow variable gives a measure of impedance to the signal-carrying variable. When a signal passes between two elements the impedances of the elements must be 'matched' in order for the expected portion of signal to be transferred between the elements, e.g. Fig 4-2 shows the desired impedances to be matched for the case of maximum signal transfer, the condition for modelling a signal flow representation of an instrument system in terms of power flow. Impedances of connected elements are joined in series for effort signal transfer and in parallel for flow signal transfer. To maximize an effort signal transfer the output impedance of the source stage should be much lower than the input impedance of the connecting stage so that the energy loss which occurs across the output impedance of the source stage is minimal. To maximize the transfer of a flow signal the output impedance of the source stage should be much higher than the input impedance of the connecting

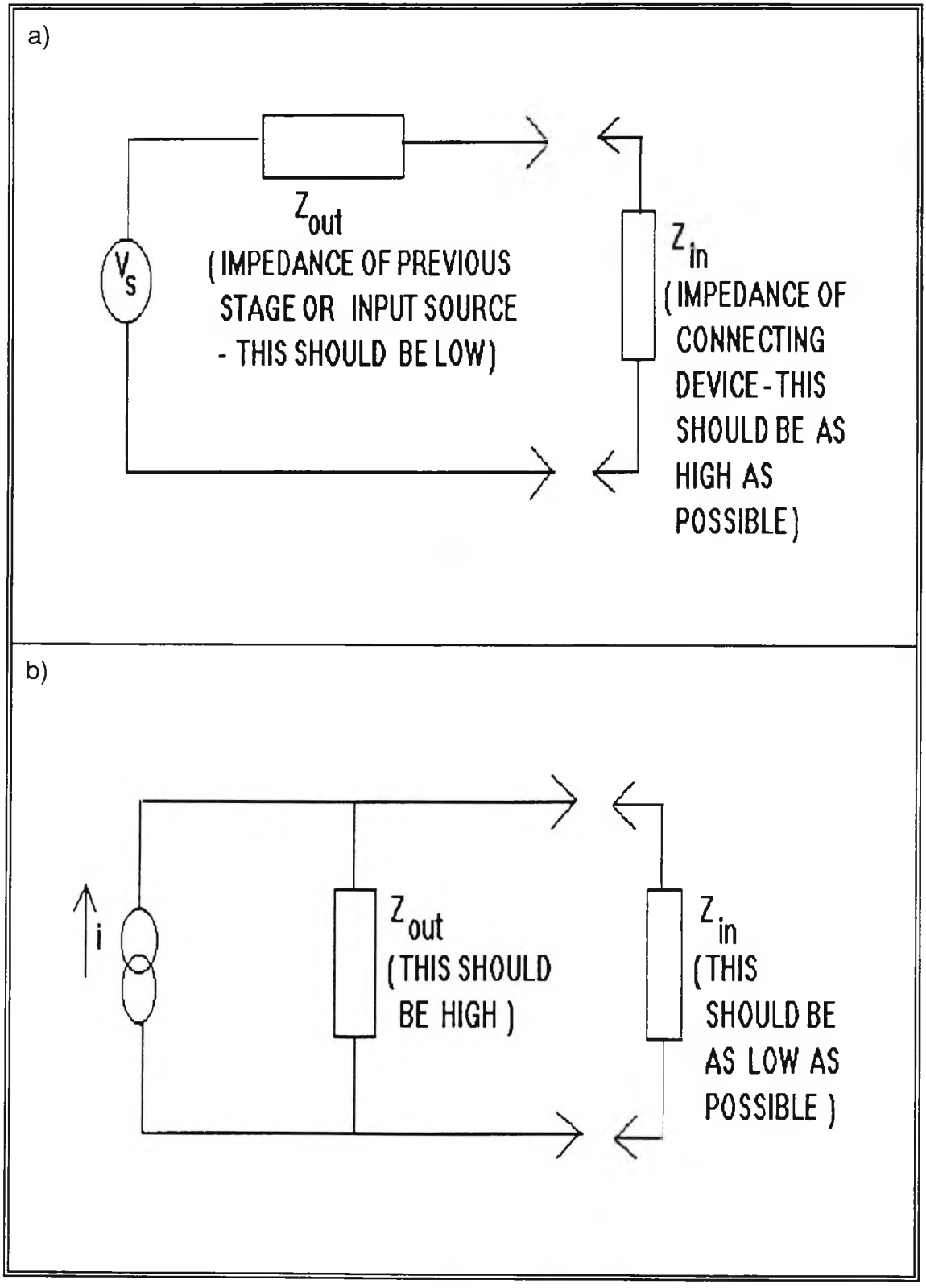



Fig 4-2 Impedance Matching for Maximum Signal Transfer

(a) For Effort Signals (b) For Flow Signals

stage so that as much of the signal as possible flows into the connecting stage.

### 4.3.2 Energy Flow in Distributed Parameter Systems

As with lumped parameter systems, the energy flow in a distributed parameter system is described in terms of a pair of associated energy rate variables, whose product is power in all energy domains apart from thermal systems, and one of which is a signal-carrying variable. This time the variables are known as the flux (denoted $\dot{\phi}$ in generalised form) and potential ($\dot{p}$ in generalised form) and the values of both vary as a function of space. Flux relates to the flow of energy at any point in space and is described by the magnitude and direction at each point. Potential is a measure of the potential between two points. For all energy domains apart from mechanical systems the value of a potential divided by the magnitude of the corresponding flux at the same point gives a measure of the impedance to a signal at that point; for mechanical systems the signal impedance is found by dividing the magnitude of the flux by the potential. The output from a distributed parameter system can be represented as a field consisting of lines of flux, formed by solving the equation $\dot{\phi}$ = constant, for many constants, and lines of equipotential, formed by solving the equation $\dot{p}$ = constant, also for many constants; both sets of lines are orthogonal to each other over the space occupied by the field. Fig 4-3 shows an example of the electric field resulting between two positive charges of 140pC separated by 127mm; the lines of flux are indicated by the solid lines and the lines of equipotential are indicated by the dashed lines. A field could also be described in terms of flux density ($\dot{\Phi}$ in generalised form) and the rate of variation of potential ($\dot{P}$ in generalised form), both of which may also vary with space. The flux density for a given area is obtained from the product of the potential gradient over the area and a specific property of the medium the field occupies, the 'acceptivity' per unit volume. A summation of the products of the flux densities in a field and the surface areas over which they extend gives the total flux. Similarly, the summation of the products of potential gradients and the lengths they occupy over a line between two points gives the potential between those points. Table 4-2 lists some commonly occurring fields and the relations for their static

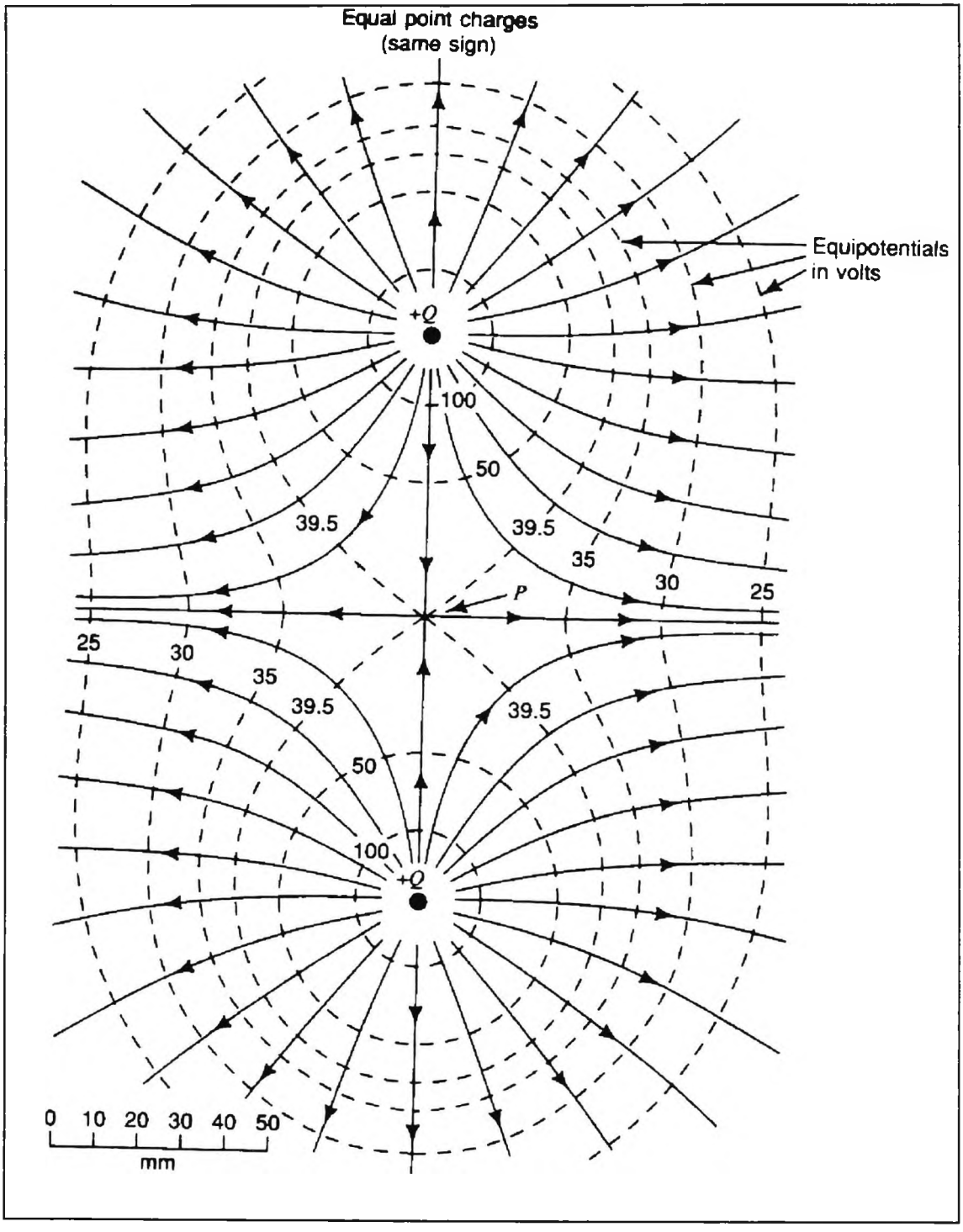



Fig 4-3 The Field Resulting Between Two Positive Electric Charges (Krauss, 1991)

| | Field Type | | | |
|---|---|---|---|---|
| Field Property | Electro-conductive | Electro-static | Magneto-static | Thermo-conductive |
| Acceptivity | Conductivity $\sigma$ [$\mho$ $m^{-1}$] | Permittivity $\varepsilon$ [F $m^{-1}$] | Permeability $\mu$ [H $m^{-1}$] | Thermal Conduct-ivity k [W $K^{-1}$ $m^{-1}$] |
| Acceptance | Conductance $G = \sigma A/l$ [$\mho$] | Capacitance $C = \varepsilon A/l$ [F] | Permeance $\Lambda = \mu A/l$ [Wb A $turn^{-1}$] | Thermal Conduct-ance $K = kA/l$ [W $K^{-1}$] |
| Potential | Electro-motive force V [V] | Potential Difference V [V] | Magneto-motive force F [A turn] | Temperature Difference $\theta$ [K] |
| Flux | Current I [A] | Electric Flux $\psi$ [C] | Magnetic Flux $\Phi$ [Wb] | Thermal Flux Q [W] |
| System Relation | $I = VG$ | $\psi = VC$ | $\Phi = F\Lambda$ | $Q = \theta K$ |
| Potential Gradient | Voltage Gradient E [V $m^{-1}$] | Potential Gradient E [V $m^{-1}$] | Magnetic Potential Gradient H [A turn $m^{-1}$] | Thermal Gradient U [K $m^{-1}$] |
| Flux Density | Current Density J [A $m^{-2}$] | Electric Flux Density D [C $m^{-2}$] | Magnetic Flux Density B [Wb $m^{-2}$] | Thermal Flux Density q [W $m^{-2}$] |
| Specific Relations | $J = \sigma E$<br>$E = -\text{grad } V$ | $D = \varepsilon E$<br>$E = -\text{grad } V$ | $B = \mu H$<br>$H = -\text{grad } F$ | $q = kU$<br>$U = -\text{grad } \theta$ |

Table 4-2 Properties of Static Fields (Vitkovitch, 1966)

properties.

The behaviour of a field depends on the properties of the medium it occupies (Vitkovitch, 1966; Ramo, Whinnery and Van Duzer, 1984; Krauss, 1991). In a homogeneous medium the properties of the medium do not vary with spatial position, but in an inhomogeneous medium they do. Media with frequency dependent properties are called dispersive. A linear medium is one in which the other field characteristics are not a function of the magnitude of the flux; if they are then the medium is nonlinear. An isotropic medium is one in which the vectors describing the other characteristics are parallel to the flux vectors; if this is not so then the medium is anisotropic. For an anisotropic medium the scaler characteristics of the field become vector quantities. If a signal is propagated via a time-varying field the static relations for the field still apply at each point in time and for a homogeneous electrical, magnetic, electromagnetic or mechanical (e.g. pressure) medium the following wave equation is obeyed:

$$\nabla^2 u = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}$$

where v is the velocity of propagation and u is some scalar or vector characteristic of the field. For thermal and fluid fields the following diffusion equation is obeyed:

$$\nabla^2 u = \frac{1}{\kappa}\frac{\partial u}{\partial t}$$

where $\kappa$ is the diffusivity of the medium. The proportion of signal transmitted across a boundary between two media is given by:

$$\tau = \frac{\phi_t}{\phi_i} = \frac{2Z_2}{Z_2 + Z_1}$$

where $\tau$ is the transmission coefficient, $\phi_t$ is the magnitude of the transmitted field, $\phi_i$ is the magnitude of the incident field, and $Z_1$ and $Z_2$ are the impedances of mediums 1 and 2. The proportion of signal reflected can be found from:

$$\rho = \frac{\phi_r}{\phi_i} = \frac{Z_2 - Z_1}{Z_2 + Z_1} \quad or \quad \rho = \tau - 1$$

where $\rho$ is the reflection coefficient and $\phi_r$ is the reflected field.

### 4.3.3 Energy Flow in Ray Approximation Systems

Energy flow in a ray approximation system is characterised by the fact that it travels in approximately straight lines in a uniform medium. This implies that light and allied electromagnetic waves at the upper end of the spectrum, e.g. ultraviolet and infrared radiation, are ray approximation signals, but there are also other radiations which satisfy these conditions, e.g. ultrasonic waves. The power in a ray approximation system is described by the radiant flux which is the rate of transfer of radiant energy from one region to another (Boyd, 1983); the corresponding signal is described by the amplitude modulated variation of the power and is denoted $\psi$ in generalised form. Alternatively, the power could be described by the radiant intensity which is the radiant flux emitted per unit solid angle, or by the irradiance which is the radiant flux per unit area. A measure of the impedance of a medium to a ray approximation signal is given by the refractive index, $\eta$, of the medium which is the velocity of a propagated signal in the medium relative to the velocity of light, i.e:

$$\eta = \frac{c}{v}$$

where c is the velocity of light and v is the velocity of propagation. At a boundary between two media a proportion of a ray approximation signal is reflected and a proportion is transmitted by refraction. The angle of reflection is the same as the angle of incidence and the angle of refraction is obtained from Snell's law which is given by:

$$\sin \theta_t = \frac{\eta_1}{\eta_2} \sin \theta_i$$

where $\theta_i$ is the angle of incidence, $\theta_t$ is the angle of refraction across the boundary and $\eta_1$ and $\eta_2$ are the refractive indices of mediums one and two. Apart from reflection and refraction, the other possible operations on signals in a ray approximation system include absorption, storage, transmission, or any combination of possible operations.

## 4.4 Instrument System Functions

Finkelstein and Watts (1983) have classified the tasks performed by instrument systems into the following functions:

- Energy storage
- Energy conversion
- Supply of power
- Interconnection
- Control of the above functions

A functional model of the signal flow through an element representing one of the above functions can be represented in the following ways:

- By a transfer function or frequency response (McGillem and Cooper, 1984) for energy conversion and energy storage.

- By a relation between signals for interconnection.

- By a description of an independent signal for power supply.

- Control functions can be described by one of the above representations and the associated amplitude or frequency signal ranges for which it applies.

The transfer function, H(s), of an element is described by the ratio of the laplace transform of an input signal applied to the element to the laplace transform of the resulting output signal, i.e.:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0}$$

where X(s) is the laplace transform of the input, Y(s) is the laplace transform of the output, $b_m \ldots b_0$ are coefficients of a polynomial representing X(s), $a_{n-1} \ldots a_0$ are coefficients of a polynomial representing Y(s), and n is greater than or equal to

m. This can be rewritten as:

$$H(s) = \frac{K(s - z_1)(s - z_2)....(s - z_n)}{(s - p_1)(s - p_2)...(s - p_m)}$$

where K is a constant known as the gain, $z_1...z_n$ are the zeroes of H(s), i.e. $H(s = z_i) = 0$ for i = 1...n, and $p_1...p_m$ are the poles of H(s), i.e. $H(s = p_j) = \infty$ for j = 1...m. The frequency response describes the variation with input signal frequency of the output signal magnitude and the phase shift. It is obtained by substituting $s = j\omega$ in H(s) and finding the resulting magnitude and phase for each frequency $\omega$; the inverse fourier transform of a frequency response is also the impulse response of a system represented in this way. If a system is modelled by a transfer function or frequency response the implication is that the system is linear, which in practice is only possible for a certain input amplitude or frequency range. A model of a practical system can be described either by the upper and lower bounds of the variation of the frequency response or by the associated transfer functions.

Instrument system functions may be further classified into the hierarchy of functions in Fig 4-4. Table 4-3, Table 4-4, Table 4-5 and Table 4-6 all show examples of components which fulfil the functions in the hierarchy for practical purposes. These components may be considered to be composed of a number of idealised elements, each representing a function in the hierarchy, but only one element having a significant effect on the input to output signal transformation; this element represents the function the component is said to fulfil. Alternatively, some components may be considered to be composed of more than one significant element, in which case they fulfil a subsystem function. The functional model of an element representing a subsystem can be described by a transfer function and may be derived from the functional models of the elements the subsystem is composed of. The signal transformations performed by the functions in the hierarchy and the associated functional models are described below for elements in their simplest form, i.e. elements with a minimum number of inputs and outputs. Each model is described by a single functional relation between the inputs and outputs. More complicated functional models are

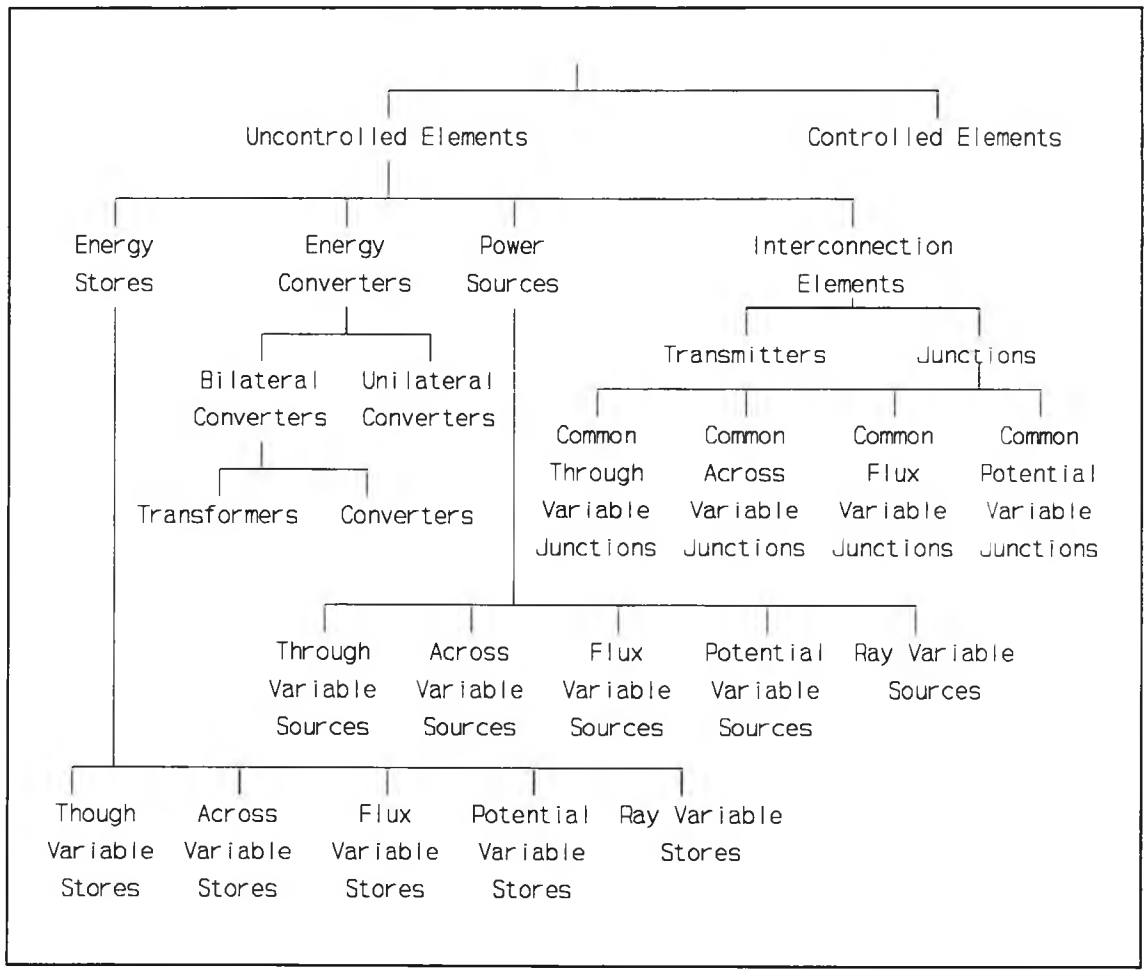



Fig 4-4 A Classification of Instrument System Functions

described by a functional relation for each set of inputs which influence an output.

### 4.4.1 Energy Stores

Energy storage elements store the energy in their input signal. Examples of components which do this are shown in Table 4-3. In the classification in Fig 4-4 energy storage functions are divided into stores for through variables, across variables, flux variables, potential variables and ray variables; these are described next.

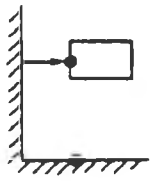
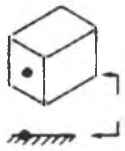
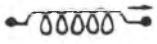
| | System | | | | |
|---|---|---|---|---|---|
| Function | Mechanical Translational | Mechanical Rotational | Electrical | Fluid Flow | Pseudo-Thermal |
| Through Variable Store |  Inertia $h = M\dot{x}$ |  Inertia $H = J\dot{\phi}$ |  Capacitance $q = Cv$ |  Capacitance $g = Cp$ |  Capacitance $Q = C\theta$ |
| Across Variable Store |  Compliance $f = Kx$ |  Compliance $T = K\phi$ |  Inductance $i = L^{-1}\lambda$ |  Inertance $\dot{g} = L^{-1}\rho$ | None |

Table 4-3 Energy Storage Components (Finkelstein and Watts,1983)

#### 4.4.1.1 Through Variable Stores

These elements store a through variable; the functional relation in this case is:

$$y = \Phi_C(\dot{x})$$

where $\Phi_C$ is the function of the through variable store. For a linear element with an across variable input signal and a through variable output signal:

$$\dot{y} = C\frac{d\dot{x}}{dt} \text{ and the transfer function is } \dot{y}(s) = sC\dot{x}(s)$$

where C is a parameter which describes the capacitance of the element to store the through variable. For a linear element with a through variable input signal

and an across variable output signal:

$$\dot{x} = \frac{1}{C}\int \dot{y}dt \text{ and the transfer function is } \dot{x}(s) = \frac{1}{sC}\dot{y}(s)$$

#### 4.4.1.2 Across Variable Stores

These elements store an across variable; the functional relation in this case is:

$$x = \Phi_L(\dot{y})$$

where $\Phi_L$ is the function of the across variable store. For a linear element with a through variable input signal and an across variable output signal:

$$\dot{x} = L\frac{d\dot{y}}{dt} \text{ and the transfer function is } \dot{x}(s) = sL\dot{y}(s)$$

where L is a parameter which describes the capacitance of the element to store the across variable. For a linear element with an across variable input signal and a through variable output signal:

$$\dot{y} = \frac{1}{L}\int \dot{x}dt \text{ and the transfer function is } \dot{y}(s) = \frac{1}{sL}\dot{x}(s)$$

#### 4.4.1.3 Flux Variable Stores

These elements store a flux variable; the functional relation in this case is:

$$\phi = \Phi_{CD}(\dot{p}) \quad or \quad \Phi = \Phi_{CD}(\dot{P})$$

where $\Phi_{CD}$ is the function of the flux variable store. For a linear element:

$$\dot{\phi} = C_D\frac{d\dot{p}}{dt} \; or \; \dot{\Phi} = C_D\frac{d\dot{P}}{dt} \; or \; \dot{p} = \frac{1}{C_D}\int \dot{\phi}dt \; or \; \dot{P} = \frac{1}{C_D}\int \dot{\Phi}dt$$

where $C_D$ is a parameter which describes the capacitance of the element to store the flux variable. The transfer function is:

$$\dot{\phi}(s) = sC_D\dot{p}(s) \quad or \quad \dot{\Phi}(s) = sC_D\dot{P}(s)$$

$$or \quad \dot{p}(s) = \frac{1}{sC_D}\dot{\phi}(s) \quad or \quad \dot{P}(s) = \frac{1}{sC_D}\dot{\Phi}(s)$$

#### 4.4.1.4 Potential Variable Stores

These elements store a potential variable; the functional relation in this case is:

$$p = \Phi_{LD}(\dot{\phi}) \quad or \quad P = \Phi_{LD}(\dot{\Phi})$$

where $\Phi_{LD}$ is the function of the potential variable store. For a linear element

$$\dot{p} = L_D \frac{d\dot{\phi}}{dt} \quad or \quad \dot{P} = L_D \frac{d\dot{\Phi}}{dt} \quad or \quad \dot{\phi} = \frac{1}{L_D}\int \dot{p}\,dt \quad or \quad \dot{\Phi} = \frac{1}{L_D}\int \dot{P}dt$$

where $L_D$ is a parameter which describes the capacitance of the element to store the potential variable. The transfer function is:

$$\dot{p}(s) = sL_D\dot{\phi}(s) \quad or \quad \dot{P}(s) = sL_D\dot{\Phi}(s)$$

$$or \quad \dot{\phi}(s) = \frac{1}{sL_D}\dot{p}(s) \quad or \quad \dot{\Phi}(s) = \frac{1}{sL_D}\dot{P}(s)$$

#### 4.4.1.5 Ray Variable Stores

These elements store a ray variable. In a ray approximation system this is done by a phosphorescent material absorbing energy and then re-emitting it. The functional relation in this case is:

$$\dot{\psi}_{OUT} = \Phi_{CR}(\dot{\psi}_{IN})$$

where $\Phi_{CR}$ is the function of the ray variable store, $\dot{\psi}_{IN}$ is the input ray variable signal and $\dot{\psi}_{OUT}$ is the output ray variable signal. This can be expressed as:

$$\dot{\psi}_{OUT}(t) = \dot{\psi}_{IN}(t - \alpha)$$

where $\alpha$ is a parameter which describes the time delay before the input signal is re-emitted. The associated transfer function is:

$$\dot{\psi}_{OUT}(s) = e^{-\alpha s}\dot{\psi}_{IN}(s)$$

### 4.4.2 Energy Converters

Energy converters transform the energy in an input signal into an output signal. Examples of components which do this are shown in Table 4-4. In the

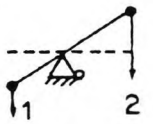
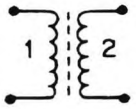
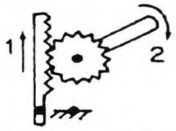
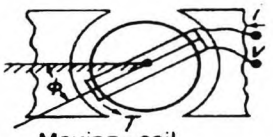
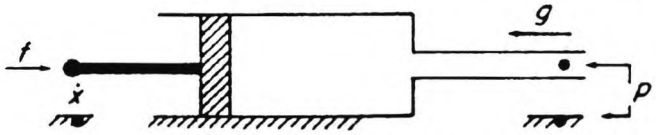
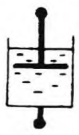
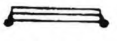
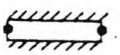
| | System | | | | |
|---|---|---|---|---|---|
| Function | Mechanical Translational | Mechanical Rotational | Electrical | Fluid Flow | Pseudo-Thermal |
| Transformer |  Lever $x_1 = nx_2$ $f_2 = -nf_1$ |  Gear $\phi_1 = n\phi_2$ $T_2 = -nT_1$ |  Transformer $v_1 = nv_2$ $i_2 = -ni_1$ | | |
| Converter |  Rack and pinion $x_1 = n\phi_2$ $T_2 + nf_1 = 0$ $T_2\dot{\phi}_2 + f_1\dot{x}_1 = 0$ |  Moving-coil electrodynamic movement $\lambda = \lambda(\phi)$ $T\dot{\phi} + vi = 0$ | | | |
| |  $g = Ax$ $p\dot{g} + f\dot{x} = 0$ Piston in cylinder | | | | |
| Unilateral Converter |  Damper friction viscous drag $f = B\dot{x}$ |  Damper friction viscous drag $T = B\dot{\phi}$ |  Resistance $v = Ri$ |  Friction  Capillary friction $p = R\dot{g}$ |  Thermal conductance $\theta = R\dot{Q}$ |

Table 4-4 Energy Conversion Components (Finkelstein and Watts, 1983)

classification in Fig 4-4 energy conversion functions are divided into bilateral converters and unilateral converters, both of which are described next.

#### 4.4.2.1 Bilateral Converters

Bilateral converters are energy converters which transform energy without storage or loss. If the input and output energies of a bilateral converter are of the same energy form then the element is known as a transformer. Those bilateral converters for which there is a change in energy form are known as converters.The functional models for the above bilateral converter elements are described next.

##### 4.4.2.1.1 Transformers

The input and output signals of a transformer are related by the following functional relation:

$$S_{OUT} = \Phi_T(S_{IN})$$

where $\Phi_T$ is the function of the transformer, $S_{IN}$ is the input signal variable and $S_{OUT}$ is the output signal variable. For a linear element:

$$S_{OUT} = NS_{IN} \quad \textit{and the transfer function is} \quad S_{OUT}(s) = NS_{IN}(s)$$

where N is a parameter which describes the amount of signal amplification or scaling performed by a bilateral converter element. The above equation applies to reflection and refraction in a ray approximation system.

##### 4.4.2.1.2 Converters

The functional relations for a converter are the same as that for a transformer, but the input and output signal variables are different due to the energy conversion.

#### 4.4.2.2 Unilateral Converters

These elements convert energy with an accompanying energy loss in the form of heat. In a lumped parameter system the functional relation is:

$$\dot{x} = \Phi_R(\dot{y}) \quad or \quad \dot{y} = \Phi_R(\dot{x})$$

where $\Phi_R$ is the function of the unilateral converter. The power balance equation is given by:

$$\dot{Q} + \dot{x}\dot{y} = 0$$

where $\dot{Q}$ is the energy loss. For a linear element:

$$\dot{x} = R\dot{y} \quad or \quad \dot{y} = \frac{1}{R}\dot{x}$$

where R is a parameter which describes the amount of dissipation associated with the element. The transfer function is:

$$\dot{x}(s) = R\dot{y}(s) \quad or \quad \dot{y}(s) = \frac{1}{R}\dot{x}(s)$$

In a distributed parameter system the functional relation is:

$$\dot{\Phi} = \Phi_R(\dot{P}) \quad or \quad \dot{P} = \Phi_R(\dot{\Phi})$$

The power balance equation is given by:

$$\dot{Q} + \dot{\Phi}\dot{P} = 0$$

For a linear element:

$$\dot{\Phi} = R\dot{P} \quad or \quad \dot{P} = \frac{1}{R}\dot{\Phi}$$

The transfer function is:

$$\dot{\Phi}(s) = R\dot{P}(s) \quad or \quad \dot{P}(s) = \frac{1}{R}\dot{\Phi}(s)$$

In a ray approximation system the functional relation is:

$$\dot{\psi}_{OUT} = \Phi_R(\dot{\psi}_{IN})$$

The power balance equation is given by:

$$\dot{Q} + W_{IN} + W_{OUT} = 0$$

where $W_{IN}$ is the power at the input port and $W_{OUT}$ is the power at the output port. For a linear element:

$$\dot{\psi}_{OUT} = R\dot{\psi}_{IN} \text{ and the transfer function is } \dot{\psi}_{OUT}(s) = R\dot{\psi}_{IN}(s)$$

which applies to absorption in a ray approximation system.

### 4.4.3 Power Sources

Power source elements deliver power at their output. Examples of components which do this are shown in Table 4-5. In the classification in Fig 4-4 power source functions are divided into sources for through variables, across variables, flux variables, potential variables and ray variables; these are described next.

#### 4.4.3.1 Through Variable Sources

A through variable source delivers a through variable signal which is independent of the corresponding across variable at the port; the functional relation is:

$$\dot{y} = \dot{y}(t)$$

#### 4.4.3.2 Across Variable Sources

An across variable source delivers an across variable signal which is independent of the corresponding through variable at the port; the functional relation is:

$$\dot{x} = \dot{x}(t)$$

#### 4.4.3.3 Flux Variable Sources

A flux variable source delivers a flux variable signal which is independent of the corresponding potential variable at the port; the functional relation is:

$$\dot{\phi} = \dot{\phi}(t) \quad or \quad \dot{\Phi} = \dot{\Phi}(t)$$

#### 4.4.3.4 Potential Variable Sources

A potential variable source delivers a potential variable signal which is independent of the corresponding flux variable at the port; the functional relation is:

$$\dot{p} = \dot{p}(t) \quad or \quad \dot{P} = \dot{P}(t)$$

#### 4.4.3.5 Ray Variable Sources

A ray variable source delivers a ray variable signal at an output port; the

| | System | | | | |
|---|---|---|---|---|---|
| Function | Mechanical Translational | Mechanical Rotational | Electrical | Fluid Flow | Pseudo-Thermal |
| Through Variable Source | Force Generator (Driver) | Torque Generator |  Current generator |  Constant-flow pump | Heat Flow Generator |
| Across Variable Source | Velocity Generator | Angular Velocity Generator |  Voltage generator |  Constant-pressure pump | Constant-Temperature Source |

Table 4-5 Power Source Components
(Finkelstein and Watts, 1983)

functional relation is:

$$\dot{\psi} = \dot{\psi}(t)$$

### 4.4.4 Interconnection Elements

Interconnection elements join the signals from other elements together. Examples of components which do this are shown in Table 4-6. In Fig 4-4 interconnection functions are divided into transmitters and junctions, both of which are described next.

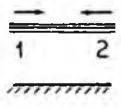
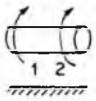
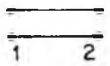
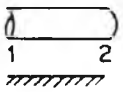
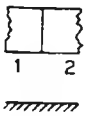
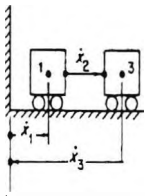
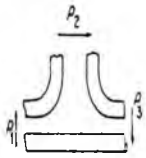
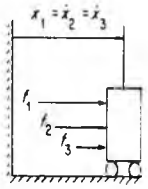
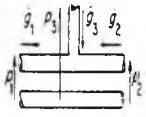
| | System | | | | |
|---|---|---|---|---|---|
| Function | Mechanical Translational | Mechanical Rotational | Electrical | Fluid Flow | Pseudo-Thermal |
| Transmitter |  Rigid rod $\dot{x}_1 = \dot{x}_2$ $f_2 = -f_1$ |  Rigid shaft $\dot{\phi}_1 = \dot{\phi}_2$ $T_2 = -T_1$ |  Lossless wire pair $v_1 = v_2$ $i_2 = -i_1$ |  Lossless pipe $p_1 = p_2$ $\dot{q}_1 = -\dot{q}_2$ |  Thermal contact $\theta_1 = \theta_2$ $\dot{Q}_1 = -\dot{Q}_2$ |
| Common Through Variable Junction |  Mechanical displacement summing at port 2 Force applied at 2 | | |  Hydraulic series interconnection | |
| Common Across Variable Junction |  Force-summing junction $f_1 + f_2 + f_3 = 0$ | | |  Hydraulic T-junction $q_1 + q_2 + q_3 = 0$ | |

Table 4-6 Interconnection Components (Finkelstein and Watts, 1983)

#### 4.4.4.1 Transmitters

A transmitter is an element which transfers the signal applied at an input port, unchanged, to an output port. For an across variable signal the functional relation

is:

$$\dot{x}_{OUT} = \dot{x}_{IN}$$

The power balance equation is:

$$\dot{x}_{IN}\dot{y}_{IN} + \dot{x}_{OUT}\dot{y}_{OUT} = 0$$

and hence:

$$\dot{y}_{IN} = -\dot{y}_{OUT}$$

Similarly, for a through variable signal:

$$\dot{y}_{OUT} = \dot{y}_{IN} \quad and \quad \dot{x}_{IN} = -\dot{x}_{OUT}$$

For a flux density signal:

$$\dot{\Phi}_{OUT} = \dot{\Phi}_{IN}$$

The power balance equation is:

$$\dot{\Phi}_{IN}\dot{P}_{IN} + \dot{\Phi}_{OUT}\dot{P}_{OUT} = 0$$

and hence:

$$\dot{P}_{IN} = -\dot{P}_{OUT}$$

Similarly, for a potential variable signal:

$$\dot{P}_{OUT} = \dot{P}_{IN} \quad and \quad \dot{\Phi}_{IN} = -\dot{\Phi}_{OUT}$$

#### 4.4.4.2 Junctions

Junctions join the signals from three elements together. They are divided into common through variable junctions, common across variable junctions, common flux variable junctions and common potential variable junctions; these are described below.

##### 4.4.4.2.1 Common Through Variable Junctions

The following equation applies in a common through variable junction:

$$\dot{y}_1 = \dot{y}_2 = \dot{y}_3$$

where $\dot{y}_1$ is the through variable at port 1, $\dot{y}_2$ is the through variable at port 2, and $\dot{y}_3$ is the through variable at port 3. The power balance equation is given by:

$$\dot{x}_1\dot{y}_1 + \dot{x}_2\dot{y}_2 + \dot{x}_3\dot{y}_3 = 0$$

where $\dot{x}_1$ is the across variable at port 1, $\dot{x}_2$ is the across variable at port 2, and $\dot{x}_3$ is the across variable at port 3. Hence:

$$\dot{x}_1 + \dot{x}_2 + \dot{x}_3 = 0 \quad or \quad -(\dot{x}_1 + \dot{x}_2) = \dot{x}_3$$

#### 4.4.4.2.2 Common Across Variable Junctions

In a common across variable junction:

$$\dot{x}_1 = \dot{x}_2 = \dot{x}_3$$

The power balance equation is:

$$\dot{x}_1\dot{y}_1 + \dot{x}_2\dot{y}_2 + \dot{x}_3\dot{y}_3 = 0$$

Hence:

$$\dot{y}_1 + \dot{y}_2 + \dot{y}_3 = 0 \quad or \quad -(\dot{y}_1 + \dot{y}_2) = \dot{y}_3$$

#### 4.4.4.2.3 Common Flux Variable Junctions

In a common flux variable junction:

$$\dot{\Phi}_1 = \dot{\Phi}_2 = \dot{\Phi}_3$$

where $\dot{\Phi}_1$ is the flux density at port 1, $\dot{\Phi}_2$ is the flux density at port 2, and $\dot{\Phi}_3$ is the flux density at port 3. The power balance equation is:

$$\dot{\Phi}_1\dot{P}_1 + \dot{\Phi}_2\dot{P}_2 + \dot{\Phi}_3\dot{P}_3 = 0$$

where $\dot{P}_1$ is the rate of variation of potential at port 1, $\dot{P}_2$ is the rate of variation of potential at port 2, and $\dot{P}_3$ is the rate of variation of potential at port 3. Hence:

$$\dot{P}_1 + \dot{P}_2 + \dot{P}_3 = 0 \quad or \quad -(\dot{P}_1 + \dot{P}_2) = \dot{P}_3$$

#### 4.4.4.2.4 Common Potential Variable Junctions

In a common potential variable junction:

$$\dot{P}_1 = \dot{P}_2 = \dot{P}_3$$

The power balance equation is:

$$\dot{\Phi}_1\dot{P}_1 + \dot{\Phi}_2\dot{P}_2 + \dot{\Phi}_3\dot{P}_3 = 0$$

Hence:

$$\dot{\Phi}_1 + \dot{\Phi}_2 + \dot{\Phi}_3 = 0 \quad or \quad -(\dot{\Phi}_1 + \dot{\Phi}_2) = \dot{\Phi}_3$$

### 4.4.5 Controlled Elements

A controlled element is one in which the value of a signal at one port can influence, without an energy conversion taking place, the functional relation between two other signals. For example, in the case of an electrical switch, which is a controlled transmitter the following functional relations apply:

$$V_{OUT} = V_{IN} \; when \; x = 0,$$

$$V_{OUT} = 0 \; when \; x < 0$$

where $V_{OUT}$ is the output voltage, $V_{IN}$ is the input voltage and x is the position of the switch. All of the elements previously described have corresponding controlled versions. In each case the functional relation for the controlled element is the same that of the uncontrolled element except for the introduction of the control variable as a parameter which influences the output.

## 4.5 Modelling of Interconnected Functions

The energy transformation performed by an instrument system may be described by an element representing the system function. This may optionally be composed of a configuration of elements, each of which represents a function in the system and may also be optionally expanded into a further configuration of elements. A model of the energy flow at a certain level of abstraction through a configuration of elements is formed by combining the models, at the same level of abstraction, of the elements in the configuration. At the signal flow level of abstraction a model of the energy flow in a functional configuration is described in terms of an overall transfer function. This can be found by algebraic manipulation of the equations relating transfer functions of the elements in the configuration. For example, in Fig 4-5 the equations are:

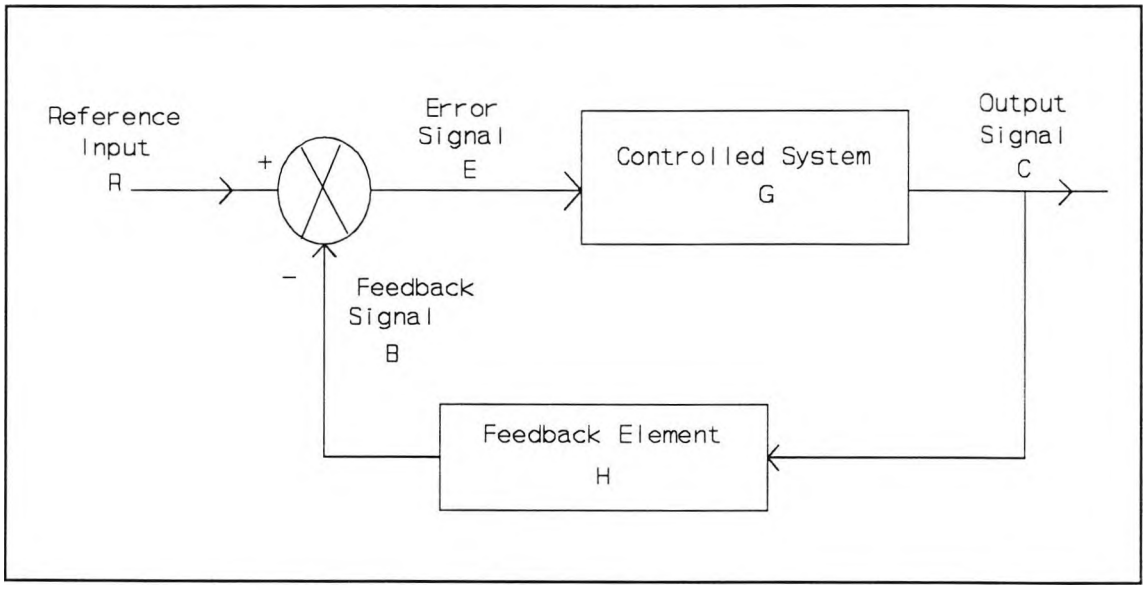



Fig 4-5 A Signal Flow Model of an Instrument System

$$C(s) = G(s)E(s)$$
$$E(s) = R(s) - B(s)$$
$$B(s) = H(s)C(s)$$

Therefore:

$$C(s) = G(s)(R(s) - H(s)C(s))$$
$$C(s) + G(s)H(s)C(s) = G(s)R(s)$$
$$C(s)(1 + G(s)H(s)) = G(s)R(s)$$

and the overall transfer function is:

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)}$$

The frequency response of the configuration can then be derived from the overall transfer function as described in section 4.4. It should be noted that if an element represents a distributed parameter system or a ray approximation system the corresponding transfer function considered here is that of the associated lumped parameter system which describes the average transfer of energy. A block

diagram such as that in Fig 4-5 can be represented in PROLOG by a number of block structures, for which the arguments are block number, block type and transmittance, i.e. the transfer function, and a number of connection structures, for which the arguments are source block number, destination block number, and sign. For example, Fig 4-5 can be described by:

```
block( 1, in, 'R' ).          connection( 1, 2, '+' ).
block( 2, sum, '_' ).         connection( 2, 3, '+ ).
block( 3, func, 'G' ).        connection( 3, 4, '+' ).
block( 4, func, 'H' ).        connection( 4, 2, '-' ).
block( 5, out, 'C' ).         connection( 3, 5, '+' ).
```

## 4.6 Automated Support for Modelling

### 4.6.1 Signal Flow Modelling Tools

MATLAB (Leonard and Levine, 1992; Saadat, 1993) is an interactive software tool for mathematical analysis which can be used for signal flow modelling. A problem is stated in MATLAB as a sequence of commands, each representing a mathematical operation, and the sequence may be parameterized if necessary. Useful sequences can be stored on disc in a file so that they are then able to be performed automatically and collections of these files are available for use in areas such as signal processing, control, system identification and optimisation. Results are displayed by a variety of different types of plotting facilities such as two-dimensional x-y plotting, logarithmic plotting, polar plots and three-dimensional perspective plots.

MATLAB is intended to be easy to use, e.g. vectors with regularly spaced elements may be generated by typing the command:

```
A = 1:4
```

which results in:

```
A = 1 2 3 4
```

and the vector A is stored for later use. All standard mathematical functions such as logarithms, sines, cosines and tangents are supported. Typing:

```
B = sin(A)
```

results in:

```
B = 0.8415    0.9093    0.1411   -0.7568
```

Control and signal processing facilities are also available and it is possible to simulate continuous or digital systems represented by transfer functions or by pole-zero plots and conversion between the representations is also possible. For example, to find the poles, zeroes and gain of the transfer function:

$$H(s) = \frac{s^3 + 11s^2 + 30s}{s^4 + 9s^3 + 45s^2 + 87s + 50}$$

the following sequence of commands is typed:

```
num = [ 1 11 30 0 ];
den = [ 1 9 45 87 50 ];
[ z, p, k ] = tf2zp( num, den )
```

num is a vector which stores the coefficients of the numerator polynomial of H(s), den is a vector which stores the coefficients of the denominator polynomial of H(s), and z, p, and k are vectors which store the zeroes, poles and gain respectively. This results in the following output:

```
z = -6.0000  -5.0000   0.0000     inf

p = -3.0000  +4.0000i
    -3.0000  -4.0000i
    -2.0000
    -1.0000

k = 1.0000
```

The impulse response of a system represented by a transfer function is obtained by typing:

```
c = impulse( num, den, t )
```

t is a vector of numbers representing time instants and c is a vector representing the impulse response at each time instant. Similarly, the step response is found by typing:

```
c = step( num, den, t )
```

and to find a frequency response of a transfer function the following is typed:

```
[ mag, phase ] = bode( num, den, w )
```

w is a vector of frequencies, mag is the vector of the associated magnitude

response at each frequency and phase is the vector which describes the phase shift at each frequency. Fig 4-6 shows a signal flow model of a system. It can be represented by the following sequence of MATLAB commands:

```
n1 = 1;       d1 = 1;       n2 = .5;      d2 = 1;
n3 = 4;       d3 = [ 1 4 ]; n4 = 1;       d4 = [ 1 2 ];
n5 = 1;       d5 = [ 1 3 ]; n6 = 2;       d6 = 1;
n7 = 5;       d7 = 1;       n8 = 1;       d8 = 1;
nblocks = 8; blkbuild;
q = [  1  0  0  0  0 % q matrix indicates the block diagram configuration
       2  1 -6 -7 -8
       3  2  0  0  0
       4  3  0  0  0
       5  4  0  0  0
       6  3  0  0  0
       7  4  0  0  0
       8  5  0  0  0 ];
iu = [ 1 ];     % input for the system
iy = [ 8 ];     % output for the system
[ A, B, C, D ] = connect( a, b, c, d, q, iu, iy )   % connect the blocks
[ num, den ] = ss2tf( A, B, C, D, 1 )      % convert to transfer function
```

This results in the output:

```
num = 0  0  0  2    den = 1.0  13.0  56.0  80.0
```

num and den are vectors storing the numerator and denominator of the overall transfer function which is therefore:

$$\frac{C(s)}{R(s)} = \frac{2}{s^3 + 13s^2 + 56s + 80}$$

SIMULINK (The MathWorks Inc, 1993) is a package for modelling and analysing dynamic systems which provides a graphical interface for the designer to specify a model and employs MATLAB for the numerical analysis required. It supports modelling of continuous and discrete linear or non-linear systems which therefore makes it suitable for modelling instrument systems. A system is specified as a diagram of interconnected blocks, each representing a model of a part of the system. These are selected and connected together by the designer via the

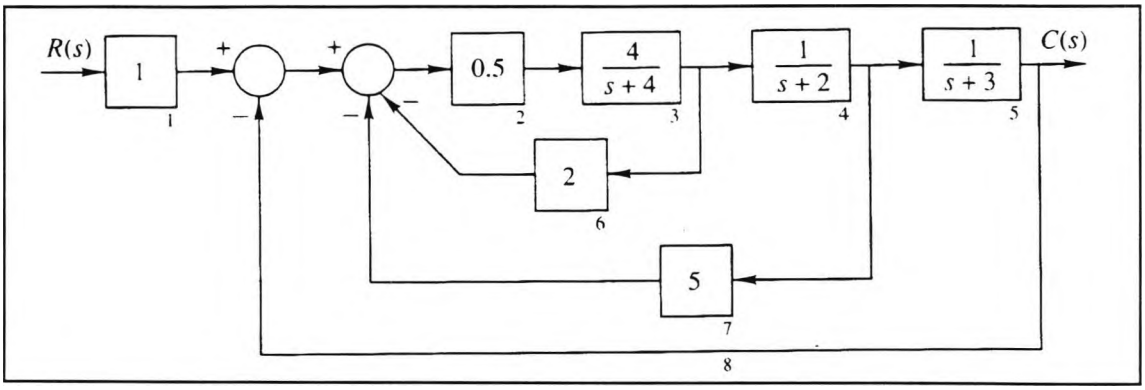



Fig 4-6 A Signal Flow Representation of a System Which can be Modelled Using MATLAB (Saadat, 1993)

interface; examples of the possible models which may be selected include step functions, sine wave generators, digital filters and transfer functions. Fig 4-7(a) shows a SIMULINK model for heat conduction in a house. The heat conduction is described by the equation:

$$\frac{dT_{in}}{dt} = \frac{1}{M_c}\left[\frac{T_{out} - T_{in}}{R_{eq}} + \dot{Q}\right]$$

where $T_{in}$ represents the indoor temperature, $T_{out}$ is the outdoor temperature, $\dot{Q}$ is the rate at which heat is added by a furnace, $R_{eq}$ is a constant related to how well the house is insulated, and $M_c$ is a constant related to the amount of air the house contains. The package also allows the designer to develop a model by first defining the subsystem models within it and these in turn may be separately defined in further detail. This enables the designer to develop a model in a top-down or bottom-up manner. The house model in Fig 4-7(a) could be defined as a single block with $\dot{Q}$ and $T_{out}$ as inputs and $T_{in}$ as an output; Fig 4-7(b) shows a SIMULINK model for a temperature control system which employs the house model in this way.

As an alternative to the block diagram form in Fig 4-5 the signal flow through an instrument system can be represented by a signal flow graph (see Fig 4-8). This is a directed graph in which the nodes represent signals and the links represent the relation between the signals, i.e. the transfer functions. Its advantage is that

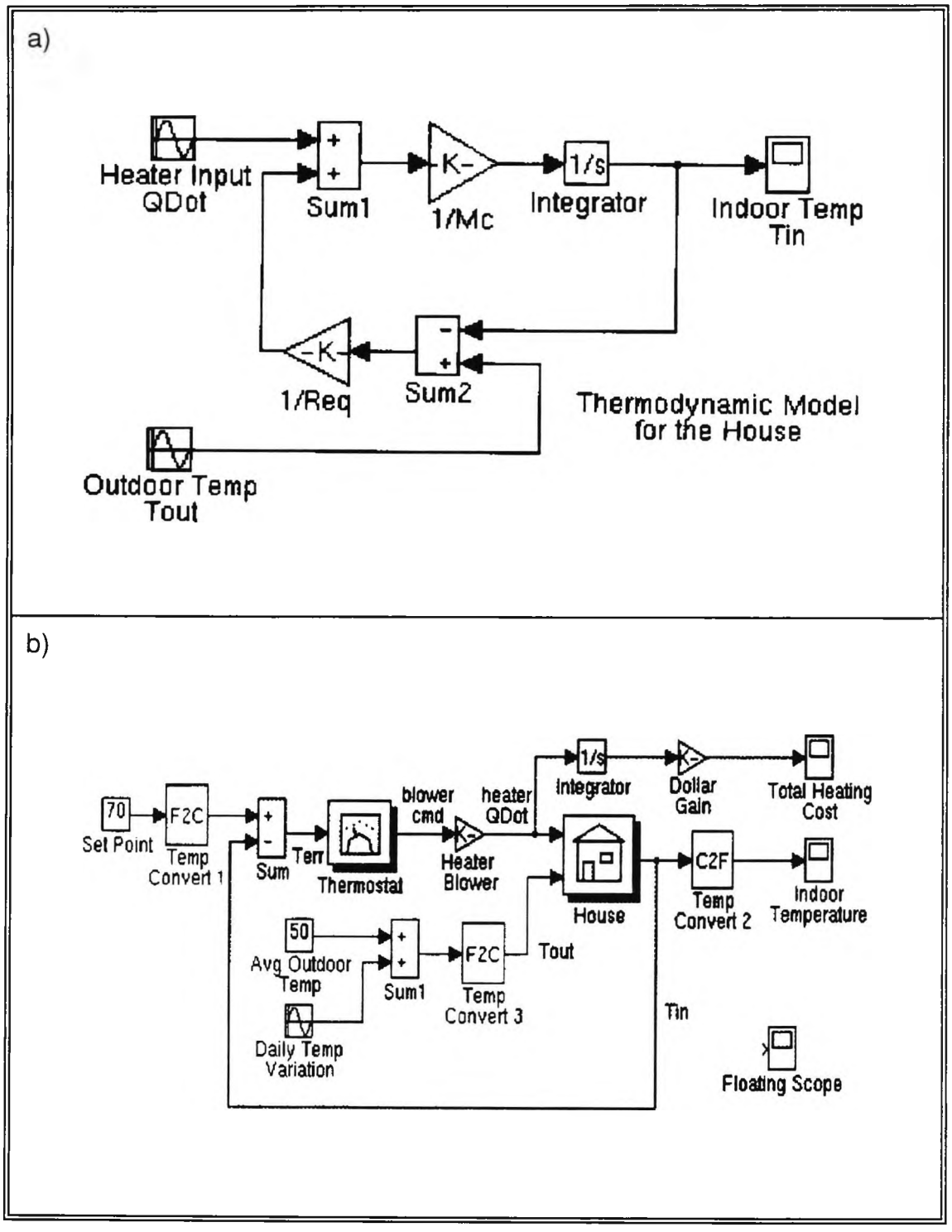



Fig 4-7 Modelling with SIMULINK (The MathWorks Inc, 1993)

(a) A Thermodynamic Model Relating the Input and Output Temperature of a House

(b) A Model for a Temperature Control System for the House Model in (a)

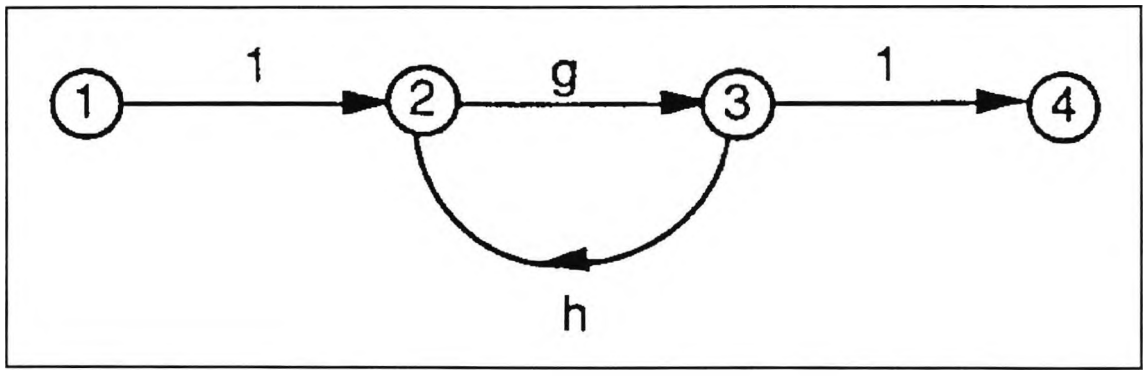



Fig 4-8 A Simple Signal Flow Graph
(Grant, Jobling and Rezvani, 1990)

the relation between two signals can quite easily be found by simplification of the graph. A signal flow graph can be represented in PROLOG by a number of connected structures, for which the arguments are the source node number, the destination node number, and the transmittance. Fig 4-8 can be described by:

```
connected( 1, 2, 1 ).
connected( 2, 3, g ).
connected( 3, 2, h ).
connected( 3, 4, 1 ).
```

Barker, Chen and Townsend (1988) describe a rule-based PROLOG program which transforms a representation of a block diagram to the equivalent signal flow graph. The description of the signal flow graph is formed by a combination of the actions performed by the rules in the program, each of which transforms the set of PROLOG structures which represent a block diagram. Jobling and Grant (1988) describe a rule-based PROLOG program for the simplification of a representation of a signal flow graph. This is done by a combination of the transformations performed by the rules in the program, each of which reduces the complexity of the signal flow graph to be simplified; Fig 4-9 shows the transformations performed by the rules. Fig 4-10 shows an example simplification of a signal flow graph. The sequence of transitions is as follows:

Fig 4-10(a) shows the initial signal flow graph.

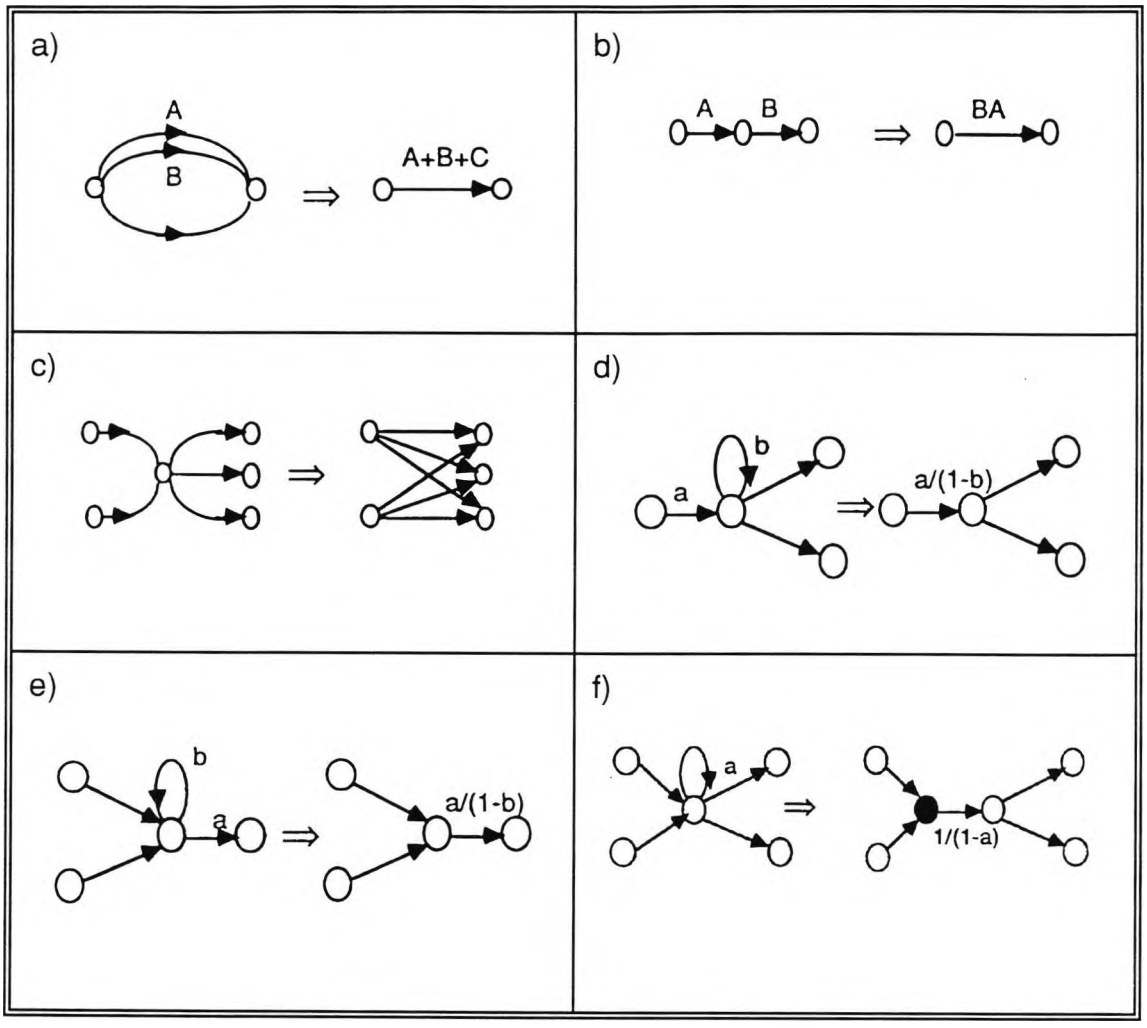



Fig 4-9 Signal Flow Graph Reduction Rules
(Jobling and Grant, 1988)

(a) Removal of Parallel Branches
(b) Combination of Branches in Series
(c) Absorption of a Node
(d) Removal of a Self-loop When There is One Non-looping Edge Into a Node
(e) Removal of a Self-loop When There is One Non-looping Edge Out From a Node
(f) Removal of a Self-loop When There are More Than Two In-edges and Out-edges

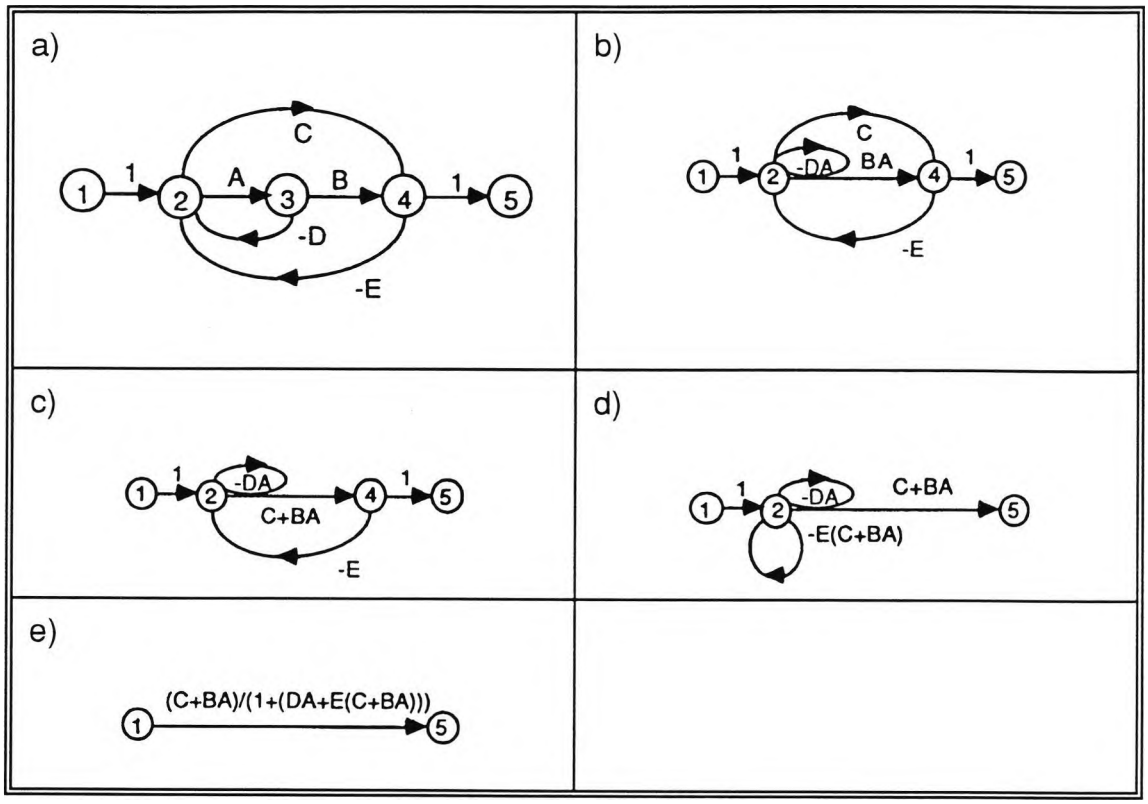



Fig 4-10 An Example Sequence of Reductions of a Signal Flow Graph (Jobling and Grant, 1988)

This is first reduced by the transformation rule in Fig 4-9(c) which is applied to remove node 3 and results in Fig 4-10(b).

The transformation rule in Fig 4-9(b) then combines the parallel edges between nodes 2 and 4 to give Fig 4-10(c).

The transformation rule in Fig 4-9(c) is then applied again to remove node 4 which leaves Fig 4-10(d).

The self-loops are then combined into a single loop using the transformation rule in Fig 4-9(a), then the loop is removed via the rule in Fig 4-9(d), and finally node 2 is absorbed using the transformation rule in Fig 4-9(b) which results in Fig 4-

10(e).

### 4.6.2 Power Flow Modelling Tools

MEDIEM (Liebner, 1981; Bailey and Abdullah, 1989) supports interactive power flow modelling and analysis of continuous dynamic systems which are represented as structure graphs. A structure graph is developed by a designer selecting the various elements in a system from a menu and connecting them together; any combination of elements forms a valid system. Each element represents a model of a single energy domain or multi-energy domain system which may be specified to be linear or non-linear; the energy domains in MEDIEM include electrical, mechanical, fluid or thermal domains. The types of elements available are shown in Fig 4-11; Table 4-7 shows examples of systems represented using structure graphs. A system described in this way is modelled by MEDIEM as a set of state equations which are used for simulation and for finding the frequency response of a system.

SPICE (Nagel, 1975) is a commonly used package for the simulation and analysis of analogue electronic circuits and can therefore be used to model the power flow in electrical and analogous systems. To define a circuit in SPICE the designer specifies a structure of nodes linked together by various electrical elements such as resistors, capacitors, inductors, voltage sources and diodes; an example circuit and the corresponding SPICE definition is shown in Fig 4-12. Each element has an associated mathematical model and a set of parameters which precisely define the characteristics of the element model, e.g. the source value of a power supply and the resistance of a resistor are both element parameters; the values of the parameters are either assigned by the designer or assume a default value. The parametric models of all elements in a circuit are then formed into a number of sets of simultaneous equations, the solutions of which provide information on either the dc, ac or transient analysis of the circuit. SPICE performs a simulation by calculating the values of the signals in a circuit at a number of discrete timepoints by using a numerical integration algorithm to convert the circuit differential equations into algebraic equations which can be

| Name | Expression | | Structure graph symbol |
|---|---|---|---|
| Across Source | AC | A – | Where A – can be:<br>AI Across Impulse<br>AS Across Step<br>AR Across Ramp |
| Through Source | TH | T – | Where T – can be<br>TI Through Impulse<br>TS Through Step<br>TR Through Ramp |
| Resistance | AC = R.TH | R | |
| Capacitance | TH = C.AC<br>(Electrical, Fluid, Thermal) | C | If Linear initial value is $AC(\phi)^{[a]}$<br>If Non-linear initial value is $AC(\phi).C'^{[a]}$ |
| | AC = C.TH<br>(Mechanical Translation, Mechanical Rotation) | C | If Linear initial value is $TH(\phi)$<br>If Non-linear initial value is $TH(\phi).C(\phi)$ |
| Inductance | AC = L.TH<br>(Electrical, Fluid, Thermal) | L | If Linear initial value is $TH(\phi)$<br>If Non-linear initial value is $TH(\phi).L(\phi)$ |
| | TH = L.AC<br>(Mechanical Translation, Mechanical Rotation) | L | If Linear initial value is $AC(\phi)$<br>If Non-linear initial value is $AC(\phi).L(\phi)$ |
| Transformer | AC1 = TF1.AC2<br>TH2 = TF2.TH1 | TF | |
| Gyrator | AC1 = GY1.TH2<br>AC2 = GY2.TH1 | | |

[a] $(\phi)$ is the value at the start of simulation.

Signal Elements

| Name | Expression | Structure Graph Symbol |
|---|---|---|
| Across to Across | AC2 = K.AC1 | AA |
| Across to Across and Integrate | AC2 = $\int$K.AC1 + AC2($\phi$) | AAI |
| Across to Through | TH2 = K.AC1 | AT |
| Across to Through and Integrate | TH2 = $\int$K.AC1 + TH2($\phi$) | ATI |
| Through to Across | AC2 = K.TH1 | TA |
| Through to Across and Integrate | AC2 = $\int$K.TH1 + AC2($\phi$) | TAI |
| Through to Through | TH2 = K.TH1 | TT |
| Through to Through and Integrate | TH2 = $\int$K.TH1 + TH2($\phi$) | TTI |

Multiport Elements

| Name | Expression | Structure Graph Symbol |
|---|---|---|
| Resistance Field | $AC_i = \sum_{j=1}^{N} R_{ij} TH_j$ | RF (Linear only) |
| Conductance Field | $TH_i = \sum_{j=1}^{N} G_{ij} AC_j$ | GF (Linear only) |
| Capacitive Field | $AC_i = \sum_{j=1}^{N} \frac{1}{C_{ij}} \int TH_j + AC_i(\phi)$ | CF (Linear only) |
| Inductive Field | $TH_i = \sum_{j=1}^{N} \frac{1}{L_{ij}} \int TH_j + TH_i(\phi)$ | LF (Linear only) |

Fig 4-11 The Elements Used by MEDIEM (Bailey and Abdullah, 1989)

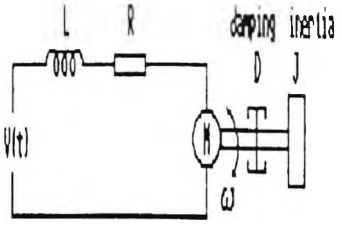
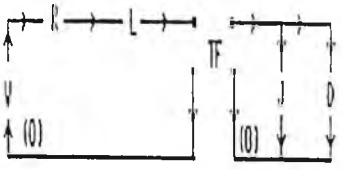
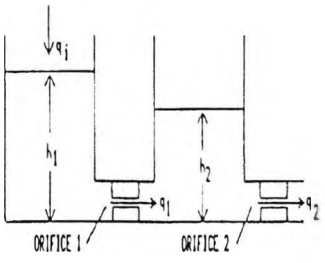
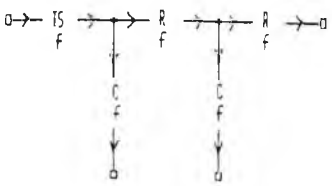
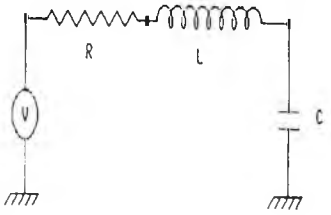
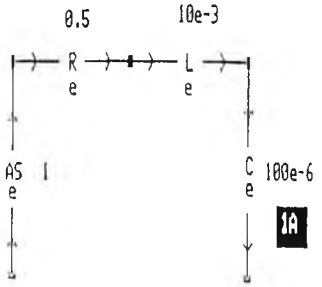
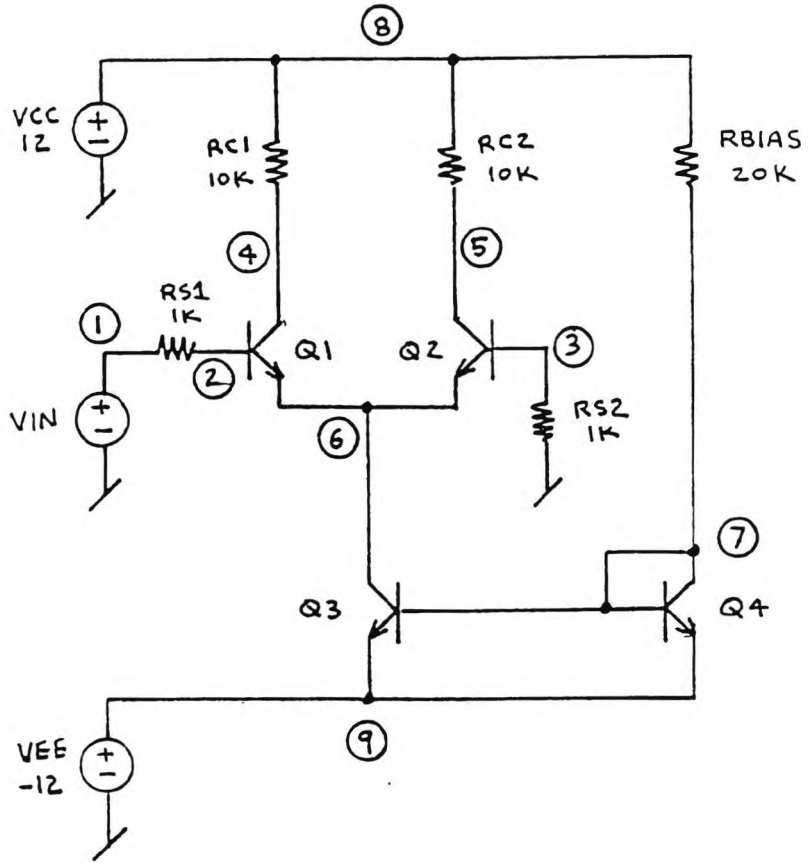
| Description of System | Schematic of System | Structure Graph |
|---|---|---|
| A dc Motor Driving an Inertial Load |   |   |
| Coupled Tanks |   |   |
| A Series RLC Circuit |   |   |

Table 4-7 System Modelling with Structure Graphs (Bailey and Abdullah, 1989)

solved at each point in time.

This type of system can be awkward to use because the number of parameters in a circuit may be very large and so it becomes difficult for the designer to know which parameters may need changing. Another disadvantage is that the models

a)




b)

```
DIFFPAIR CKT - SIMPLE DIFFERENTIAL PAIR
VIN 1 0 SIN(0 0.1 5MEG 5NS) AC 1
VCC 8 0 12
VEE 9 0 -12
Q1 4 2 6 QNL
Q2 5 3 6 QNL
RS1 1 2 1K
RS2 3 0 1K
RC1 4 8 10K
RC2 5 8 10K
Q3 6 7 9 QNL
Q4 7 7 9 QNL
RBIAS 7 8 20K
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+  CJC=2PF VA=50)
.END
```

Fig 4-12 Analogue Circuit Definition with SPICE (Nagel, 1975)

(a) Schematic for a Differential Pair Circuit
(b) Equivalent SPICE Description

are only approximations and therefore only give approximate results - although for SPICE the accuracy is very good for most purposes. Also running an analogue simulation can take a large amount of computer time. Despite these problems tools such as SPICE are used as a starting point for circuit development.

## 4.7 Conclusions

Classifications have been given of the energy which may be transformed by an instrument system and the functions which may be performed by such a system, and a survey of the automated tools for signal flow and power flow modelling has also been carried out. The knowledge based system to be described in the next two chapters is required to verify the correctness of the specification of signal flow at each level in the functional decomposition of an instrument system. This chapter has described the possible functions which may appear in a functional decomposition and the associated signal transformations. The verification can be done by checking that the variation in frequency response associated with each function in a decomposition is larger than the variation in frequency response associated with the configuration of functions the function is composed of. The variation in frequency response of a configuration of functions is obtained by finding the highest and lowest frequency responses of the configuration when the constraints on the frequency responses of each element in the configuration are taken into account. A power flow modelling system could be used to calculate the frequency response of a configuration at a signal flow level of abstraction if the impedances between elements are matched for maximum signal transfer as shown in Fig 4-2. Therefore a power flow modelling scheme could also be used to find the variation in frequency response by calculating each possible frequency response of a configuration, but a signal flow modelling system would be more efficient. This would first convert a functional configuration to the equivalent signal flow graph, which could then be simplified in the way shown in Fig 4-10. The result is a symbolic description of the overall transfer function of the configuration. This will be of the form of one symbolic expression divided by another; the symbols in the expressions represent element transfer functions.

The resulting upper bound of the frequency response can be obtained by the following actions:

- Substitute in the numerator expression the actual transfer functions which correspond to the upper bounds of the frequency responses of the elements in the numerator.

- Substitute in the denominator expression the actual transfer functions which correspond to the lower bounds of the frequency responses of the elements in the denominator.

- Use MATLAB to calculate the resulting transfer function and convert this to a frequency response.

The resulting lower bound of the frequency response can be obtained by the following actions:

- Substitute in the numerator expression the actual transfer functions which correspond to the lower bounds of the frequency responses of the elements in the numerator.

- Substitute in the denominator expression the actual transfer functions which correspond to the upper bounds of the frequency responses of the elements in the denominator.

- Use MATLAB to calculate the resulting transfer function and convert this to a frequency response.

Once these two frequency responses are obtained they can then be checked against the expected variation in frequency response.

For example, after the reduction of the signal flow graph in Fig 4-10(a) to that in

Fig 4-10(e) the overall simplified transfer function is:

$$\frac{C(s) + B(s)A(s)}{1 + ( D(s)A(s) + E(s)( C(s) + B(s)A(s) ))}$$

The numerator is C(s) + B(s)A(s). The denominator is 1 + ( D(s)A(s) + E(s)( C(s) + B(s)A(s) )). Suppose A(s), B(s), C(s), D(s) and E(s) are all transfer functions associated with converter functions which all have a conversion factor of between 1 to 10. The upper bound of the numerator is: 10 + 10∗10 = 110, and the lower bound of the denominator is: 1 + (1∗1 + 1∗(1 + 1∗1)) = 4. Therefore the resulting transfer function associated with the upper bound of the frequency response is: 110/4 = 27.5. In this case the upper bound of the frequency response is 27.5 for all frequencies. The lower bound of the numerator is: 1 + 1∗1 = 2, and the upper bound of the denominator is: 10 + (10∗10 + 10∗(10 + 10∗10)) = 1210. Therefore the resulting transfer function associated with the lower bound of the frequency response is: 2/1210 = 0.00165. In this case the lower bound of the frequency response is 0.00165 for all frequencies. This variation in frequency response can now be compared with that expected.

# CHAPTER 5

# INSTRUMENT SYSTEM FUNCTIONAL DATA STORAGE

## 5.1 Introduction

This chapter describes the necessary functional information and the associated data structures stored in a knowledge based system (KBS) which aids the generation of design concepts for instrument systems. It starts with an outline of the structure of the system and then the next sections describe in detail the representation in the KBS of a functional specification of an instrument system, its decomposition and solution, and the structure of the different knowledge bases of the system.

## 5.2 The Structure of the Knowledge Based System

The KBS is composed of the following elements:

- The designer interacts with the system through an interface which should be graphical to facilitate the interaction. Stored descriptions of the functional specification of an instrument system, the functional decompositions of the specification, and the design concepts which meet the specification should be available to the designer and easy to update via the interface.

- Knowledge bases storing known functional configurations, laws of physical effects, and solution characteristics of lumped parameter, distributed parameter, and ray approximation system functions; this knowledge should be organised

according to several classification schemes.

- Mechanisms to locate all information stored.

- Software to support the generation of design concepts from a functional specification or its functional decomposition. Design concept generation requires the designer interface, utilities to store and retrieve the information in the knowledge bases and an inference engine to synthesize the design concepts (see Fig 5-1).

- A symbol manipulation and numerical analysis package which implements the functional modelling scheme described in chapter four.

It is intended that the software should be written in PROLOG or a similar language and the knowledge representation scheme should be based on frames. PROLOG (Clocksin and Mellish, 1984; Bratko, 1990) was chosen because it is suitable to represent frames with (Cuarado and Cuarado, 1986) and it allows problems to be expressed at a high level easily, and thus enables prototypes to be built quickly; it also has good symbol manipulation facilities which are needed when forming a description of a system using symbolic descriptions of components within the system. Frames were chosen because of their flexibility and ease of use for knowledge representation. It is also intended that the system should be integrated into an environment which supports the design process such as the viewpoint oriented systems engineering (VOSE) framework (Finkelstein, Nuseibeh, Finkelstein and Huang, 1992). This environment supports the integration of development strategies from a number of different perspectives; each 'viewpoint' describes a partial specification of a problem domain presented in a particular notation and developed using a particular strategy.

## 5.3 Representation of the Functional Decomposition of an Instrument System

The functional specification of an instrument system may be represented as

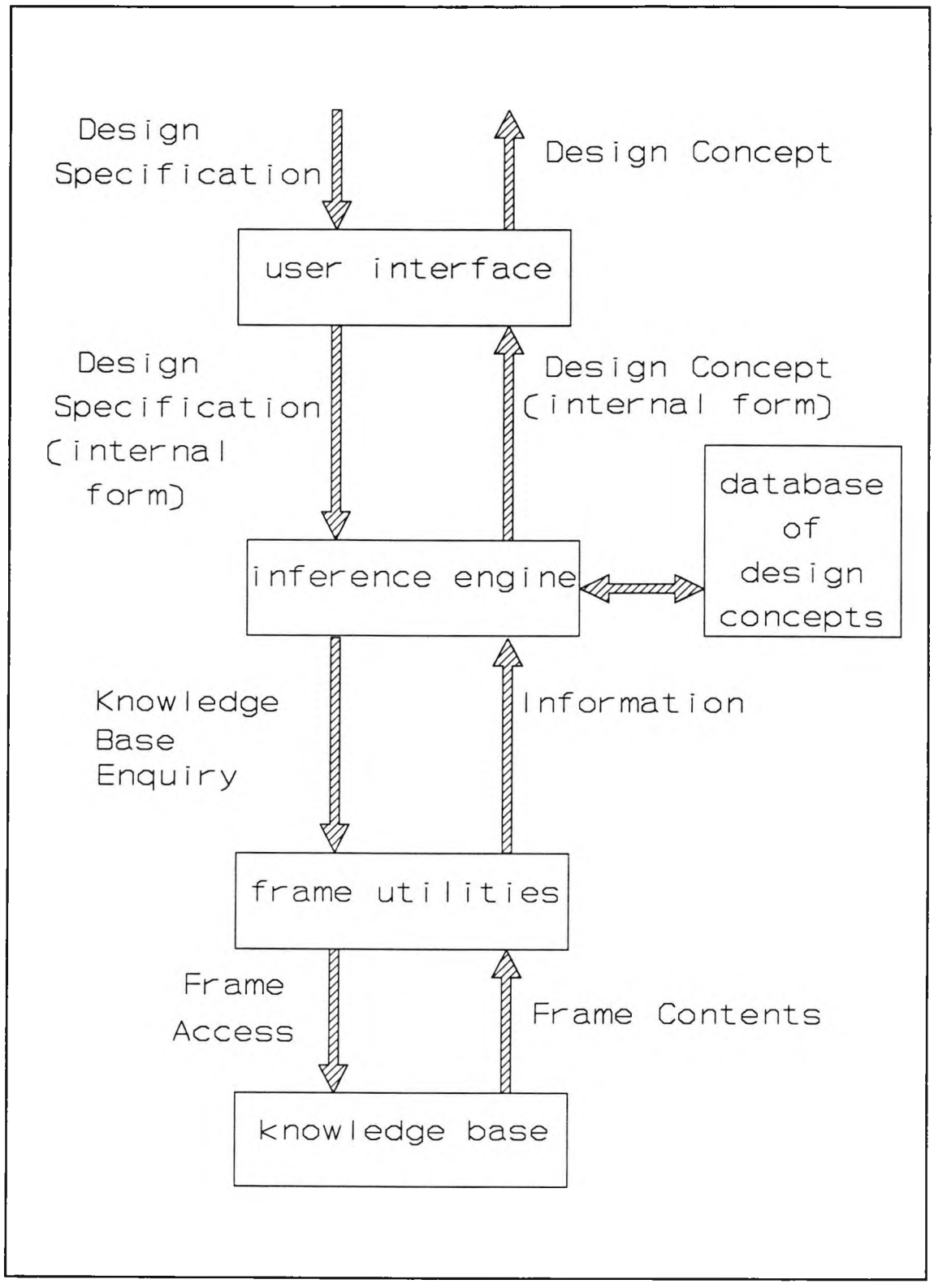
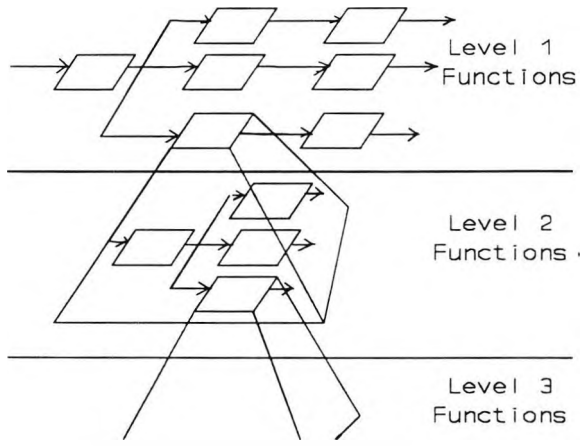



Fig 5-1 Design Concept Generation Using a Knowledge Based System

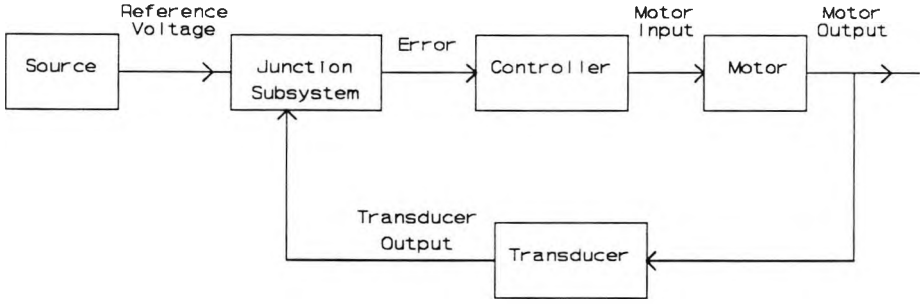
either a single function or a configuration of functions; each function is described in terms of its input, output and control signals, the functions they are connected to, and the desired constraints on the parameters of the function, e.g. the frequency response of a subsystem may have to be greater than 1 dB down for frequencies above 1 kHz. The requirements can be specified at any number of levels of abstraction as any function can be further defined to be composed of a configuration of functions, each of which may also be defined in more detail (see Fig 5-2(a)). A solution to a functional specification can also be described in the same way, but this time the constraints on the parameters of the functions are solution constraints. Fig 5-2(b) shows an example of how this form of representation is expressed graphically for one level of abstraction. Each block in the diagram is given a symbolic name and denotes a function defined at the level of abstraction the diagram represents. A connection between blocks represents the transfer of a signal between two functions in the system and all signals are also given a symbolic name. Functional constraints could also be displayed in the diagram, in which case the parameters and constraints associated with each function would be displayed.

The above graphical representation can also be stored in a PROLOG database as a collection of declarations which represent facts about the functions in the specification, the constraints on the parameters of the functions, the connections between functions via signal transfer, and any associated graphical information. Each declaration contains a number and its associated data which is related to a function or port in a configuration. A PROLOG description of Fig 5-2(b) is shown in Fig 5-2(c). Information on each function is described by a PROLOG structure of the form *function(<function number>, <function type>, <function name>, <input ports>, <output ports>, <control ports>, <x and y coordinates of the function>)*, e.g. function( 1, converter, "Heat Sensor", [1, 2], [3, 4], [5], [50.0, 30.0]) represents function number 1 which is a converter function named "Heat Sensor" with input ports 1 and 2, output ports 3 and 4, control port 5, and coordinates 50.0, 30.0; the same port number may be referred to by structures representing functions at different levels of abstraction. A function with no control

a)




b)




c)

```
function( 1, source,                "Source",              [], [ 1 ], [], _),
function( 2, subsystem,             "Junction Subsystem", [ 2,3 ], [ 4 ], [], _),
function( 3, controller_subsystem,  "Controller",          [ 5 ], [ 6 ], [], _),
function( 4, motor_subsystem,       "Motor",               [ 7 ], [ 8 ], [], _),
function( 5, converter,             "Transducer",          [ 9 ], [ 10 ], [], _),
connected( [ 1 ], [ 2 ], _),  connected( [ 4 ], [ 5 ], _),  connected( [ 6 ], [ 7 ], _),
connected( [ 8 ], [ 9 ], _),  connected( [ 10 ], [ 3 ], _),
signal( 1, voltage, "Reference Voltage" ),   signal( 2, voltage, "Reference Voltage" ),
signal( 3, voltage, "Transducer Output" ),   signal( 4, voltage, "Error" ),
signal( 5, voltage, "Error" ),               signal( 6, voltage, "Motor Input" ),
signal( 7, voltage, "Motor Input" ),         signal( 8, torque,  "Motor Output" ),
signal( 9, torque,  "Motor Output" ),        signal( 10, voltage, "Transducer Output").
```

Fig 5-2 Graphical Representation of the Functional Decomposition of an Instrument System

(a) A System Defined at Multiples of Abstraction
(b) An Example Description at One Level of Abstraction
(c) A PROLOG Representation of (b)

ports is represented by a symbol for the empty list, [], in the place where the control ports are expected. If a function is decomposed into a configuration of functions this is represented by a structure of the form *composed(<function number>, <list of function numbers in associated configuration>)*, e.g. composed( 1, [2, 3, 4]). If a function is part of a specification, but is not intended to be realised by the designer then this is represented by a structure of the form *dont_realise(<function number>)*. If the KBS attempts to generate a solution for a function and it found to be unrealisable based on the information in the knowledge base of solution characteristics this is represented by a structure of the form *unrealisable(<function number>)*. The signals at each port associated with a function are described by structures of the form *signal(<port number>, <signal name>, <symbolic name of signal>)* or *signal(<list of port numbers>, <list of associated signal names>, <list of associated symbolic names>)*, e.g. signal( 1, temperature, "Input") or signal( [1, 2], [force, velocity], ["Control", "Output"]). Connections between ports are represented by structures of the form *connected(<source port number>, <receiver port number>, <list of coordinates of line segments>)* or *connected(<list of source port numbers>, <associated receiver port numbers list>, <associated line segments list>)*.

The constraints on the parameters of each function are represented by a structure of the form *parameter(<parameter name>, <associated port numbers>, <associated function number>, <constraints on the parameter>)*, e.g. parameter( amplitude, [1], 1, at_least( 20, 40 )) describes an amplitude range of at least between 20 to 40 units of the signal at port 1 of function 1 and parameter( amplitude, [1], 1, at_most( 20, 40 )) describes an amplitude range of at most between 20 to 40 units of the signal at port 1 of function 1. In addition to these constraints if the constraints on a parameter vary with the amplitude or frequency of a signal this is represented by a structure of the form *condition(<parameter constraints>, <associated amplitude or frequency range>)* for each amplitude or frequency range which the constraints on the parameter vary with, e.g. condition( parameter( 'R', [1, 2], 1, at_most( 20, 40 )), parameter( amplitude, [1], 1, at_most( 5, 10 ))) represents constraints related to function 1 with a value of

between 20 to 40 for the parameter R, the amount of energy loss associated with a unilateral converter, which are valid for an amplitude range of between 5 to 10 units of the signal at port 1. Table 5-1 shows the naming conventions for parameters. Storage of constraints on the following parameters is included:

- The amplitude, frequency and power at the ports associated with all functions.

- The impedances at the ports associated with lumped parameter and distributed parameter system functions.

- The parameters described in chapter four which directly influence the input to output transformation of energy conversion and energy storage functions, e.g. the amplification or scaling factor (N) associated with a bilateral converter. The constraints on these parameters also define the frequency response and impulse response of the associated functions. The variation of frequency response, impulse response and step response, and the associated transfer functions do not need to be stored as they can be calculated from the constraints on these parameters.

- The accuracy associated with the parameters which influence the input to output transformation of energy conversion and energy storage functions.

- The frequency response of subsystem functions. This is stored for a number of frequency ranges which cover the entire frequency range to be considered; each range is of an equal, system-defined length. Each structure associated with a frequency response stores the variation of frequency response for a specific frequency range. The variation of impulse response and step response and the associated transfer functions for a subsystem do not need to be stored as they can be calculated from the variation of the frequency response.

- The parameters related to step response, e.g. the percentage overshoot, peak value, peak time, rise time, and settling time.

| | System Description | | |
|---|---|---|---|
| Parameter | Lumped Parameter System | Distributed Parameter System | Ray Approximation System |
| Frequency Response | response( $f_1$, $f_2$ ) | response( $x_1$, $y_1$, $x_2$, $y_2$, $f_1$, $f_2$ ) | response( $\theta_1$, $\theta_2$, $f_1$, $f_2$ ) |
| Ray Approximation System Parameters | | | < parameter name > |
| All Other Parameters | < parameter name > | < parameter name > ( $x_1$, $y_1$, $x_2$, $y_2$ ) | < parameter name > ( $\theta_1$, $\theta_2$ ) |

Table 5-1 The Naming Conventions for Parameters in the KBS

- The height, width, cross-sectional area, and other spatial measures associated with distributed parameter and ray approximation system functions.

- For distributed parameter system functions the above parameters vary with space and are stored for a number of cross-sectional areas which cover the entire area to be considered; each area is of an equal, system-defined size. Each structure associated with a distributed parameter representation stores the variation of a parameter for a certain area of space. A lumped parameter system description is also included for distributed parameter system functions.

- For ray approximation system functions the above parameters vary with angle and are stored for a number of ranges of angles which cover the entire range to be considered; each range is of an equal, system-defined length. Each structure associated with a ray approximation function stores the variation of a parameter for a specific range. A lumped parameter system description is also included for ray approximation system functions.

- The parameters associated with ray approximation system functions, e.g. directivity, the ratio of the maximum power to the average power, and half-power beam width, the angle between two half power levels in a main beam.

It should be noted that this list is a summary of the main parameters and information on other parameters could be added to the knowledge based system if necessary.

A modification that could be made to the above representation is the introduction of storage for functions which have the same signal transformation specification as another function which is already defined and so this does not need to be defined again; each of these functions is represented by a structure of the form *function(<function number>, instance(<referred function number>), <function name>, <input ports>, <output ports>, <control ports>, <x and y coordinates of the function>)*, e.g. function( 2, instance(1), "Heat Sensor 2", [6, 7], [8, 9], [10], [10.0, 20.0]) represents a function 2 which is defined in the same way as function 1, but is named "Heat Sensor 2", has input ports 6 and 7, output ports 9, control port 10, and coordinates 10.0, 20.0. Another modification is the partitioning of the representation into a number of different contexts or 'views' as defined by Logan, Millington and Smithers (1991); each view is a collection of assumptions representing the constraints on a particular decomposition. Solutions could be generated by the KBS for a particular view which is represented by a structure of the form *view(<view number>, <view name>, <list of functions in the view>)* and a number of structures of the form *assumption(<view number>, <associated assumption>)*; the associated assumptions are represented by structures of the

form already described for functional decompositions. A view may also inherit the assumptions in another view and so a tree of related views can be built up; each view inherits the assumptions in a 'universal view' which stores the representation of a functional structure when a view is not specified. If one view inherits the assumptions in another this is represented by a structure of the form *sub_view(<parent view number>, <child view number>)*. A conflict between assumptions in related views can be resolved by giving priority to the assumption at the lower level in the tree when this view is the one solutions are to be generated for.

Alterations made to a functional structure can be reflected in the corresponding PROLOG representation by updating the contents of the database. A declaration is added to the database by use of the *assert* PROLOG predicate and removed by use of the *retract* predicate. The following example interaction with a PROLOG system illustrates how these predicates are used; the output from the system is in bold. To ask the system if any information on converter functions named "Heat Sensor" is held the following is typed:

```
?- function( Number, converter, "Heat Sensor",
              Input, Output, Control, Position ).
no
```

and the system indicates it is not in the database. To add information on a converter function named "Heat Sensor" the following is typed:

```
?- assert( function( 1, converter, "Heat Sensor",
                     [1, 2], [3, 4], [5], [50.0, 30.0])).
yes
```

and the same query now shows it is present:

```
?- function( Number, converter, "Heat Sensor",
              Input, Output, Control, Position ).
    Number = 1,
    Input = [ 1, 2 ],
    Output = [ 3, 4 ],
    Control = [ 5 ],
    position = [ 50.0, 30.0 ]
```

If this declaration needs to be removed from the database the following is typed:

```
?- retract( function( Number, converter, "Heat Sensor",
                      Input, Output, Control, Position )).
yes
```

and the same enquiry:

```
?- function( Number, converter, "Heat Sensor",
             Input, Output, Control, Position ).
no
```

now shows it has been removed.

The current representation of a specification or solution can be saved by writing each declaration to a file so that the contents of the file can then be read at a later time and added to the database again. When the contents of a file of declarations are added to the database the function and port numbers in each new declaration in the database will be altered from those in the file to avoid confusion with the information related to the other function and port numbers already in the database, but the information in the original representation in the file is still preserved.

## 5.4 The Structure of the Knowledge Base of Known Functional Configurations

The knowledge base of known functional configurations consists of a number of frames, each containing configurations for a particular function with specific input, output and control signals. The frames are named according to the functions and associated signals they contain information on, e.g. a frame named *CONVERTER-[FORCE]-[TEMPERATURE]-[]* would contain functional configurations for converter functions with an input of force, an output of temperature and no control signals. They hold a number of *'EXAMPLE'* slots, each containing a reference to a file holding a description of a functional configuration which is represented in the same way as that already described for functional structures, but constraints on parameters are not included. Some of the functional configurations stored may represent the same information at different levels of abstraction. A frame may also hold a number of *'SUBFUNCTION'* slots, each containing a reference to another frame which stores functional configurations for subfunctions with the

same associated signals. An example of a typical part of this knowledge base is shown in Fig 5-3.

## 5.5 The Structure of the Knowledge Base of Solution Characteristics

The knowledge base of solution characteristics contains information on the functions performed by an instrument system, sets of components which perform the functions, and functional specifications for individual components. This is stored at three corresponding levels of abstraction, each represented by a structure of frames. The overall structure is intended to enable easy access, updating and expansion to the knowledge base whenever necessary. A description of the frame structure and contents at each level of abstraction is given below.

At the highest level of abstraction information related to the functions performed by an instrument system is stored. An example of a typical part of the frame structure at this level is shown in Fig 5-4. Each frame at this level represents a function and is given the same name as the function. If a function a frame represents can be divided into subfunctions, e.g. a store can be divided into an effort store and a flow store, this is indicated by each *'SUBFUNCTION'* slot in the frame containing a reference to a frame representing the subfunction. All frames at this level of abstraction contain a number of *'CONFIGURATION'* slots, each of which holds the following:

- A reference to a file containing a description, represented in the same way as a functional specification or solution but without constraints on parameters, which links specific signals to the ports of a representation of a function associated with the frame. This configuration is associated with the *'CONFIGURATION'* slot. The representation also contains details of the functions compatible at each port. Information on each compatible function is stored in a structure of the form *compatible(<port number>, <associated function number>, <port number of associated function>)*.

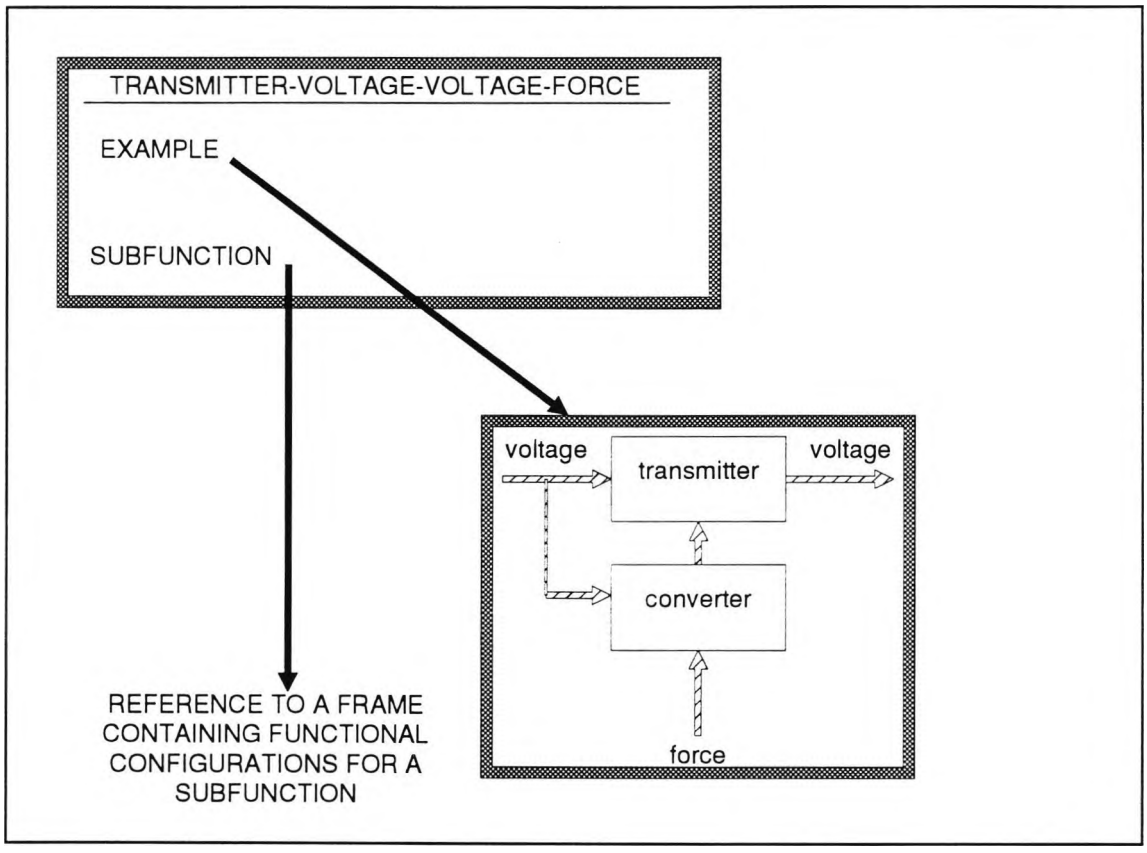



Fig 5-3 An Example Portion of the Knowledge Base of Known Functional Configurations

- A reference to another frame containing a number of *'SOLUTION'* slots, each of which contains parameter name and an associated a reference to a frame at the next level of abstraction. These frames contain information, classified according to values of the parameter, on the solution characteristics of a set of components, each of which is a realisation of the configuration associated with the *'CONFIGURATION'* slot.

The frames at the highest level of abstraction are organised into the same hierarchy of functions in Fig 4-3, but with more sub-divisions, which if necessary may be further sub-divided, and a hierarchy of subsystem functions is also included; Fig 5-5 shows example hierarchies of controlled and uncontrolled converter functions. Using the information at this level the compatibility between

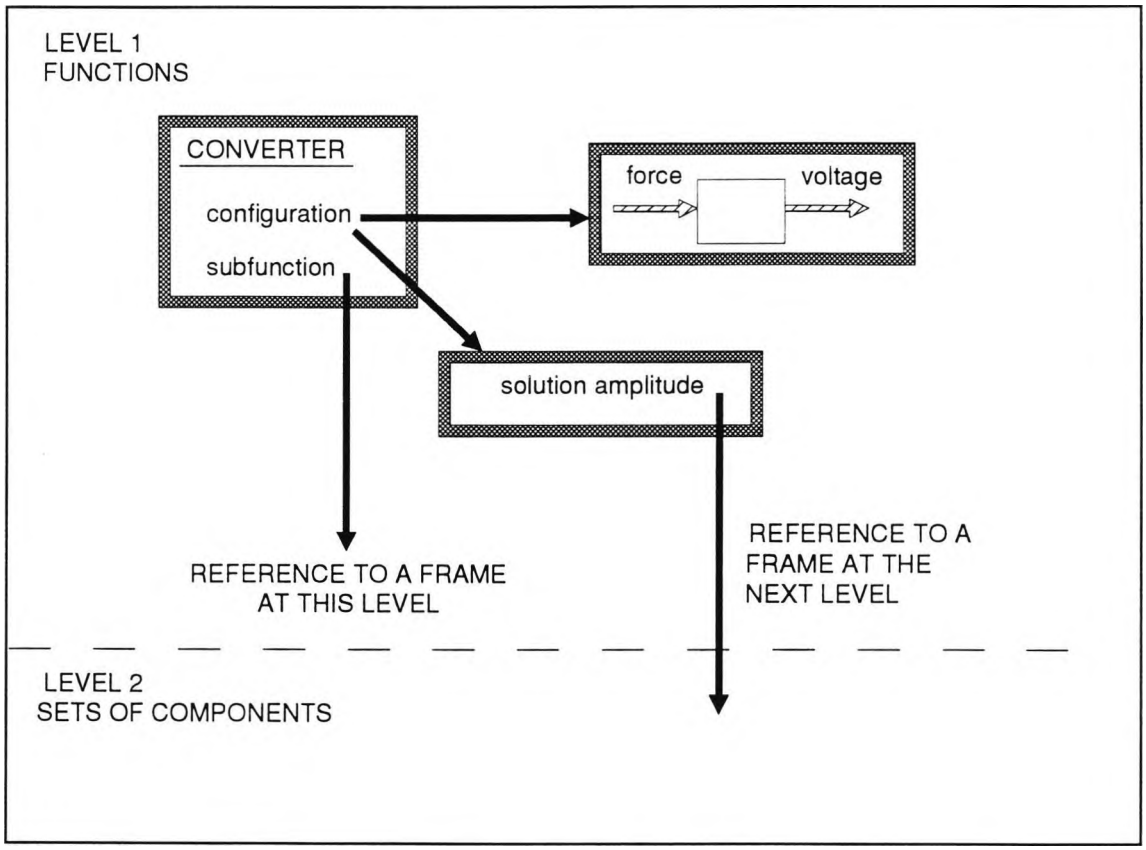



Fig 5-4 An Example Portion of the Frame Structure Related to Functions in the Knowledge Base of Solution Characteristics

realisations of functions can be checked and it can be determined if a particular function with specific input, output and control signals is realisable by components in the knowledge base.

At the next level of abstraction in the knowledge base information is stored on sets of components which perform the functions at the highest level of abstraction. Each frame at this level represents a set of components identified by the function they perform, the associated signals, a parameter, known as the classification parameter, the port numbers related to the classification parameter in a functional configuration at the highest level of abstraction in the knowledge base, and the range of the classification parameter for the set of components; this also defines the symbolic name for the frame, e.g. a frame representing a

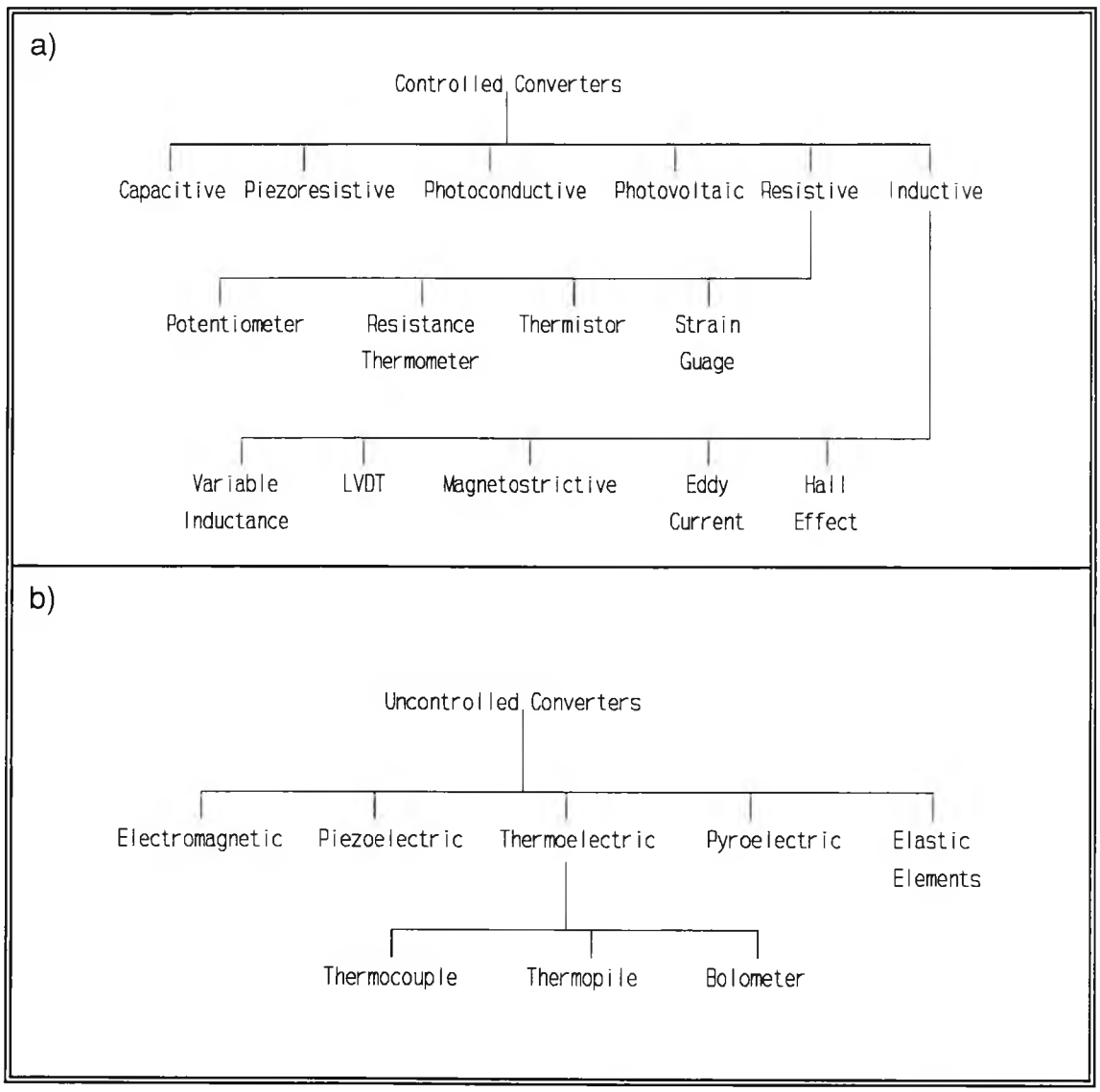



Fig 5-5 A Classification of Converter Functions (Bentley, 1988)

(a) Controlled Converters
(b) Uncontrolled Converters

set of LVDT components with an amplitude range of between 0 to 600 units of the signal associated with port one would be named *LVDT-[FORCE]-[VOLTAGE]-[]-AMPLITUDE-[1]-[0,600]*. The ranges stored are either from the lowest possible value of a parameter to a certain value or from a certain value to the highest possible value. The following slots are present in a frame at this level:

- A *'CHARACTERISTICS'* slot which contains a reference to a file storing a representation of the solution characteristics of the set of components the frame represents. This is obtained from the union of the solution characteristics considered of the components in the set, e.g. if two components perform the same function and one has a frequency response of greater than 1 dB down over a certain frequency range and the other component has a frequency response of greater than 2 dB down over the same frequency range the union of these characteristics is a frequency response of greater than 1 dB down over the frequency range. These solution characteristics do not include the variation of characteristics with signals of particular amplitude or frequency ranges.

- A number of *'SUBSET'* slots may be present, each of which contains a range associated with the classification parameter and a reference to a frame at the same level of abstraction; the frame represents a subset of components with the associated range for the classification parameter.

- A number of *'COMPONENT'* slots, each of which contains a reference to a frame at the next level of abstraction; each of these frames contains detailed information on a component which is in the set of components represented by the frame containing the reference to it.

This level of abstraction is organised into a tree structure of frames to enable efficient access to frames representing a set of components in the knowledge base which perform a particular function and have a classification parameter within a certain range. A typical structure for six components is shown in Fig 5-6. In this example the components have a parameter value of between 0 to 600 which is subdivided into subsets having values between 0 to 100, 200, 300, 400, 500, and 600 and between 600 to 100, 200, 300, 400, and 500. Sedgewick (1983) discusses the relevant data structures for organising the information, e.g. binary tree or balanced tree representations. Using the information at this level of abstraction the set of components which perform a function can be found when the required constraints on the signals and parameters of the function are

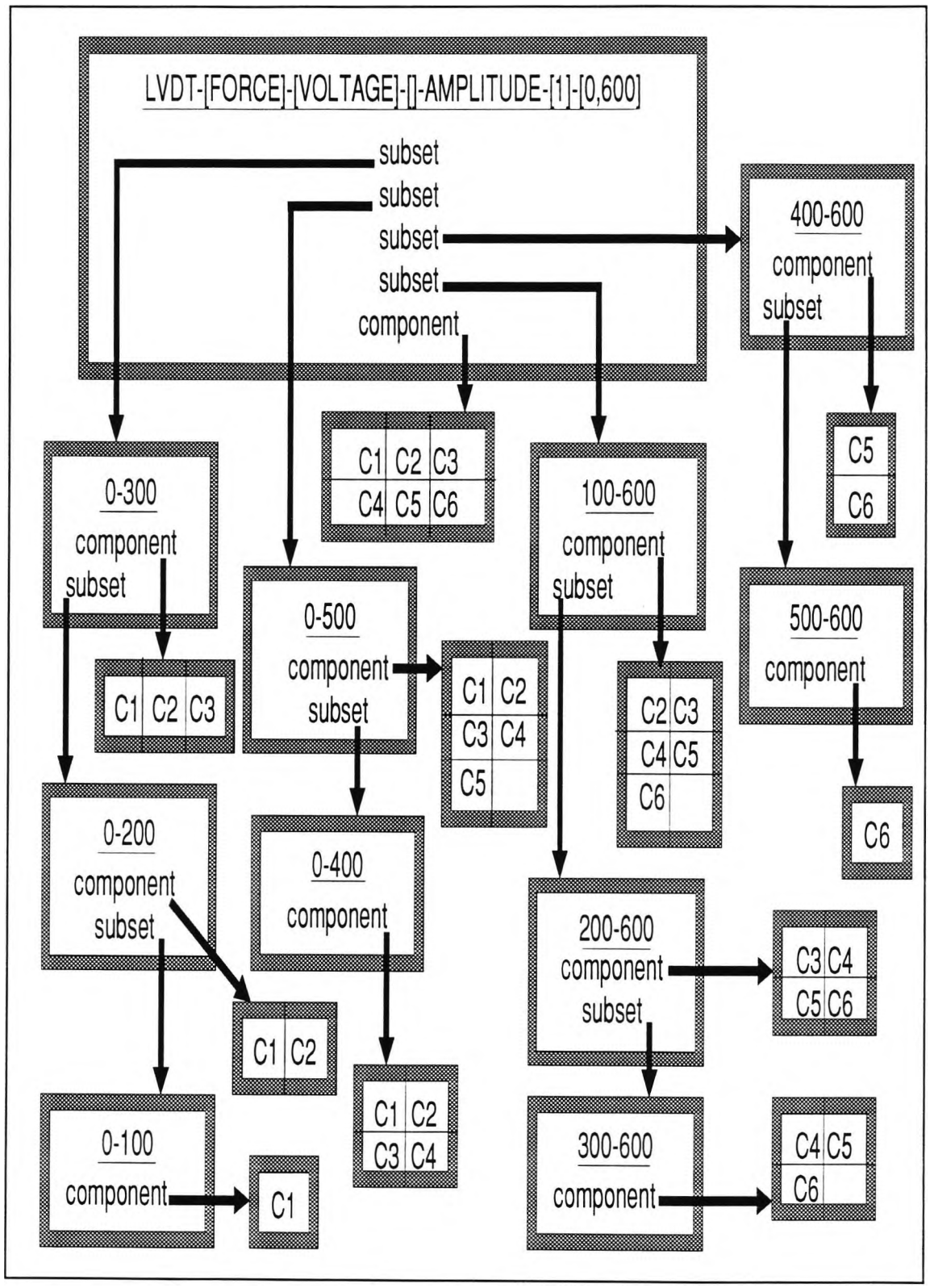



Fig 5-6 An Example Organisation of a Set of Components in the Knowledge Base of Solution Characteristics

given. The procedure to do this is as follows:

1) Choose a parameter of the function which is constrained.

2) If the parameter is constrained to be within a certain range find the set of components which perform the function, require the appropriate signal to operate, and have a value of greater than or equal to the lower bound and less than or equal to the higher bound for the parameter. For example, in Fig 5-6 the set of components with a parameter value of between 150 to 410 is determined by finding the set of components with a value of greater than or equal to 150 for the parameter, which is {C3, C4, C5, C6}, and intersecting this set with the set of components having a value of less than or equal to 410 for the parameter, which is {C1, C2, C3, C4}; the resulting set is {C3, C4}.

3) If the parameter is constrained to be at least a certain range find the set of components which perform the function, require the appropriate signals to operate, and have a value of less than or equal to the lower bound and greater than or equal to higher bound for the parameter. For example, in Fig 5-6 the set of components with a parameter value of at least between 150 to 410 is determined by finding the set of components with a value of less than or equal to 150 for the parameter, which is {C1}, and intersecting this set with the set of components having a value of greater than or equal to 410 for the parameter, which is {C5, C6}; the resulting set is {}, i.e. no components exist in the knowledge base which satisfy the constraints for the parameter.

4) Repeat steps 1), 2) and 3) for all constrained parameters of the function.

5) The set of components which perform the function and satisfy the constraints is obtained by the intersection of the sets of components found. If this results in an empty set then no solutions are present in the knowledge base. The associated design concept is obtained from the union of the solution characteristics of the selected set of components.

At the lowest level of abstraction in the knowledge base each frame represents an individual component and is named according to a component reference number. Each frame at this level holds a *'CHARACTERISTICS'* slot which contains a reference to a file storing a complete representation of the solution characteristics for the component, including the variation of all characteristics with signals of specific amplitude or frequency ranges. Using the information at this level of abstraction it is possible to find a set of components which have desired characteristics over specific amplitude or frequency ranges by checking the characteristics of each component in a group of components selected using the information at the above level of abstraction. As with information retrieval at the previous level of abstraction, the associated design concept is obtained from the union of the solution characteristics of the selected set of components.

## 5.6 The Structure of the Knowledge Base of Physical Effects

The knowledge base of laws of physical effects is organised according to the energy domains associated with the input, output and control signals in each law stored and so represents the three-dimensional 'sensor effect cube' described in Middlehoek and Hoogerwerf (1986) (see Fig 5-7). The energy domains considered here are magnetic, electrical, thermal, mechanical and radiant energy domains. A list of laws classified in this way is available in van Putten (1988); Table 5-2 lists some of them. The knowledge base stores information at two levels of abstraction as shown in Fig 5-8. At the top level a single frame named *SIGNALS* is present which holds a number of *'TRANSITION'* slots. Each of these contains the names of an input, output and control signal variable and a reference to a frame at the next level which contains information on the laws for that transition. At the next level each frame represents a set of laws for a transition between two energy domains which is subject to the influence of another energy domain as a modulator. Each frame is named according to the energy domains in the transition it contains information on, e.g. a frame named *MECHANICAL-ELECTRICAL-MAGNETIC* contains information on laws for a transition from mechanical to electrical domains with a magnetic signal as the controlling variable. A frame at this level contains a number of *'LAW'* slots, each containing

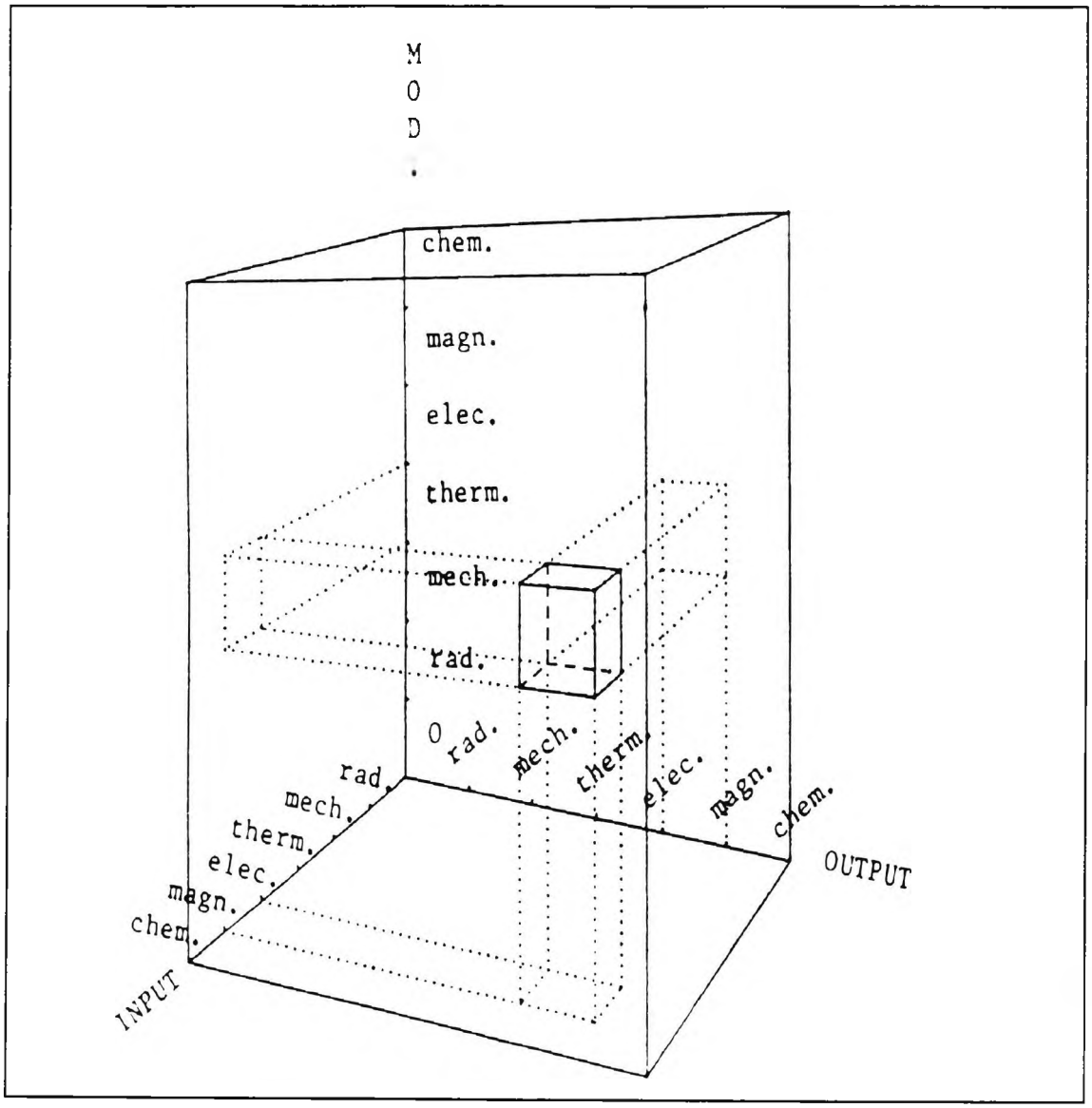



Fig 5-7 The 'sensor effect cube'
(Middlehoek and Hoogerwerf, 1986)

| Name of Effect | Equation | Description |
|---|---|---|
| Photovoltaic | $v = f( I )$ | A voltage is generated by incident radiation at the junction of two dissimilar materials |
| Radiation Heating | $\theta = f( I )$ | The increase of temperature of a material by incident radiation |
| Electroluminescence | $I = f( \psi )$ | The illuminating excitation of a material due to an alternating electrical field |
| Incandescence | $I = f( i )$ | The emission of radiation by thermal movement of atoms activated by an electric current |
| piezoelectric | $i = f( f )$ | The generation of a surface charge due to a mechanical force |
| piezoresistance | $v = f( f, i )$ | The change in conductivity in semiconductors due to a mechanical force |
| Thermoelastic Effect | $v = f( f, \theta )$ | The generation of a voltage between two regions in a metal due to mechanical strain and their temperature difference |
| Thermoelectric Seebeck | $i = f( \theta )$ | The generation of an electric current in a closed loop of two dissimilar conductors by different junction temperatures |
| Peltier Effect | $\theta = f( i )$ | The generation of a temperature difference between two junctions when a current passes through it |
| Thermoconductivity | $v = f( \theta )$ | The change in conductivity due to temperature |
| Magnetostriction | $f = f( B )$ | The mechanical deformation of a material by a magnetic field |
| Hall Effect | $\psi = f( B, i )$ | The generation of an electric field in a conductor due to a current and a magnetic field which are mutually perpendicular |
| Magnetoresistance | $v = f( B, i )$ | The change in resistivity of a material by a magnetic field |
| Electromagnetic | $B = f( i )$ | The change in magnetization due to an electric current |

Table 5-2 Examples of Known Physical Effects (van Putten, 1988)

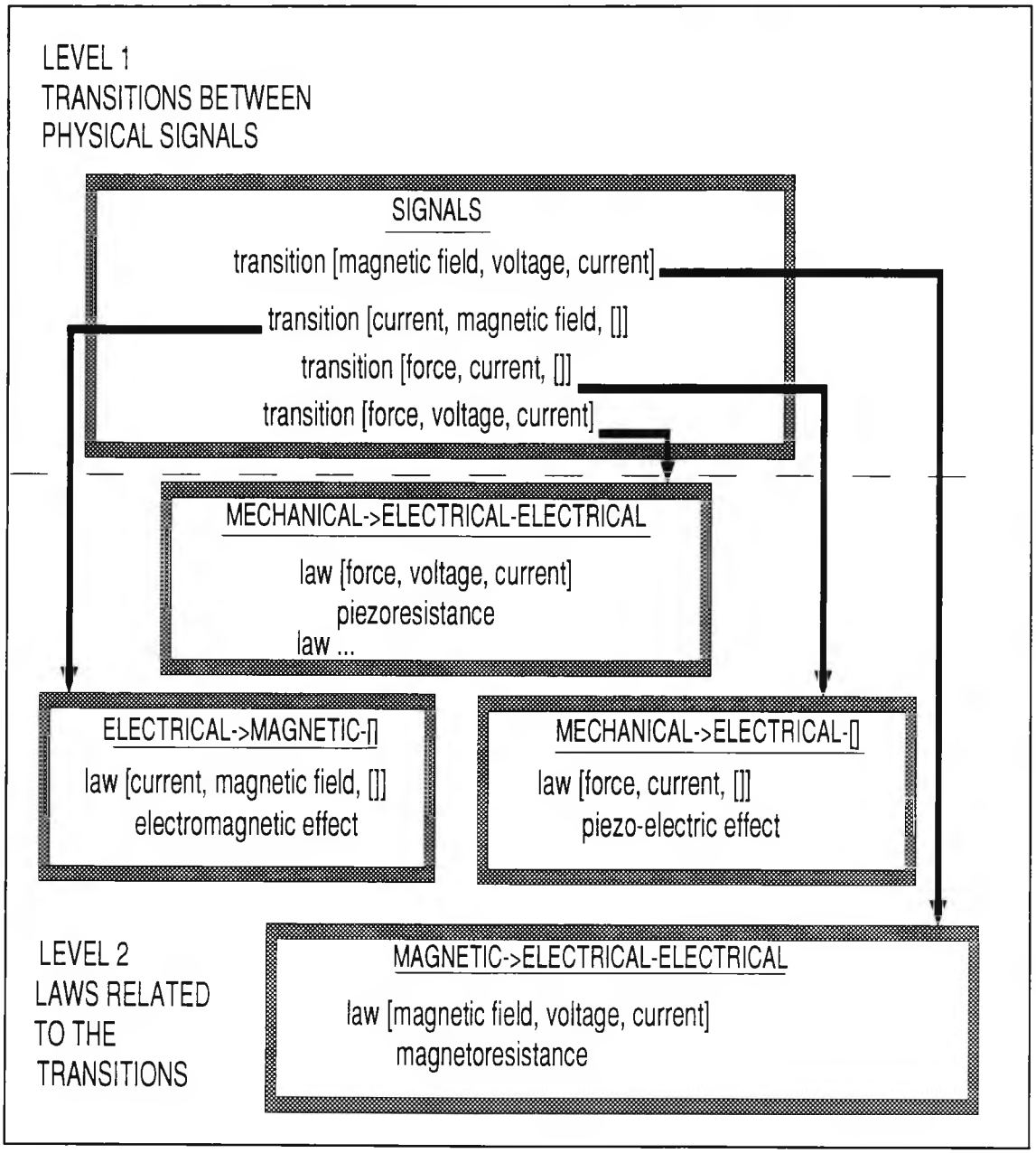



Fig 5-8 An Example Portion of the Knowledge Base of Physical Effects

the equation for the physical law associated with the transition and a description of the transition.

## 5.7 Summary

A detailed description has been given of the organisation of the knowledge based system, including the data structures used in the knowledge bases of known functional configurations, solution characteristics, and laws of physical effects. The next chapter describes how the information stored can be used to aid the generation of design concepts for instrument systems.

# CHAPTER 6

# INSTRUMENT SYSTEM DESIGN CONCEPT GENERATION

## 6.1 Introduction

This chapter describes the support facilities of a knowledge based system for the generation of design concepts for instrument systems and how they can be used. It starts with a description of the techniques for design concept generation to be supported and the conditions for design concept generation with the KBS, and then the next sections describe in detail the implementation of the support facilities of the system, the interface to the system for a designer, and an example of the possible use of the system.

## 6.2 Design Concept Generation Techniques

A review of design methodology, including the techniques for design concept generation is given by Finkelstein and Finkelstein (1983) and provides a foundation for work on an automated aid for design concept generation. The techniques for design concept generation were divided into convergent methods and divergent methods. A convergent method focuses a design solution space and therefore helps a designer to arrive at a more specific solution, e.g. by systematic search of a design catalogue. A divergent method expands a design solution space which leads to a more abstract solution. This can be done, for example, by use of novel group processes such as brainstorming or synectics (Pahl and Beitz, 1988) which encourage a creative approach to the problem. Finkelstein and Finkelstein (1983) list the main characteristics of divergent

methods as:

- expansion of the field from which the ideas are drawn
- elimination of preconceptions in the definition of the problem
- elimination of consideration of authority and convention
- neglect of constraints at the outset
- limited sequential elaboration
- separation of idea generation from evaluation

This thesis concentrates on support for the following methods of design concept generation:

(i) *USE OF EXISTING CONCEPTS*

Design concepts can be developed from a suitable selection and combination of known solutions. Each function in a specification can either be replaced by some form of realisation or decomposed into a structure of functions, each with an associated realisation.

(ii) *SYSTEMATIC TRANSFORMATION OF EXISTING CONCEPTS*

New design concepts may be generated from existing solutions by altering part of a functional structure or by considering alternative physical realisations for particular functions.

(iii) *USE OF ANALOGIES*

Systematic transformation of analogous functional descriptions of physical systems provides another way of generating design concepts. The analogy used here is that between physical signals: all through signals are known to be analogous, as are all across signals, all flux signals, all potential signals and all ray signals.

(iv) *USE OF FIRST PRINCIPLES*

A systematic examination of the physical laws can be used to suggest other

ways of realising a function. For example, Hooke's law (force = stiffness * displacement) suggests a number of ways to use a spring as a transducer: a force may be applied to the spring resulting in a displacement or a displacement may be applied resulting in a force in the spring; in both cases the stiffness can have a controlling action; alternatively, if the stiffness is variable it could also be converted to a force or displacement.

## 6.3 Support for Design Concept Generation

A design concept for a function can be generated immediately by the KBS accessing the knowledge base of solution characteristics in two types of situation which are:

- When the knowledge base contains solution characteristics for a function which meet the specification for the function.

- When the knowledge base contains solution characteristics for a configuration of functions which meet the specification for the function. This can be verified by an implementation of the functional modelling scheme described in chapter four.

In the second case the problem of finding a valid configuration can be tackled by a top-down or bottom-up approach or a combination of both methods. Top-down design involves decomposing a problem until it consists of a number of realisable subproblems; this does not always work as some problems may not be able to be solved by decomposition. Bottom-up design involves combining specific solutions to subproblems until the overall problem is solved; similarly, this is not always successful. If the KBS cannot produce a design concept the designer must either alter the specification or look elsewhere for solutions and if they are found update the knowledge base with the information gained. The implementation of support facilities to aid design concept generation is described next.

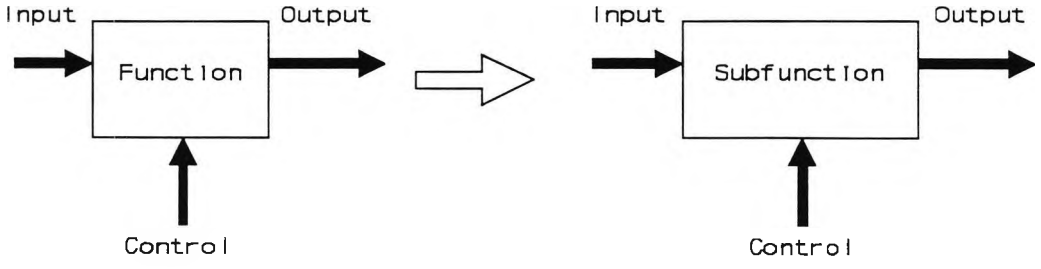
### 6.3.1 Support for Design Concept Generation by use of Existing Concepts

To aid design concept generation by use of existing concepts the knowledge base of known functional configurations can be accessed and the retrieved configurations used as a starting point for the current functional configuration to be produced. The designer may invoke the KBS to generate functional configurations for a particular function with an associated set of input, output and control signals, which may or may not be completely specified. For example, configurations may be generated for converter functions with an input of force, an output of voltage and an unspecified control signal. In this case the KBS first generates a number of control signals, and then for each control signal the functional configurations associated with the specified signals and the control signal are generated; another option is for the designer to specify a number of different signals which may be used as the control signal and then invoke the KBS to generate the appropriate functional configurations. Alternatively, a designer may manually search through the functional configurations stored and select a suitable one for further development.

### 6.3.2 Support for Design Concept Generation by Systematic Transformation of Existing Concepts

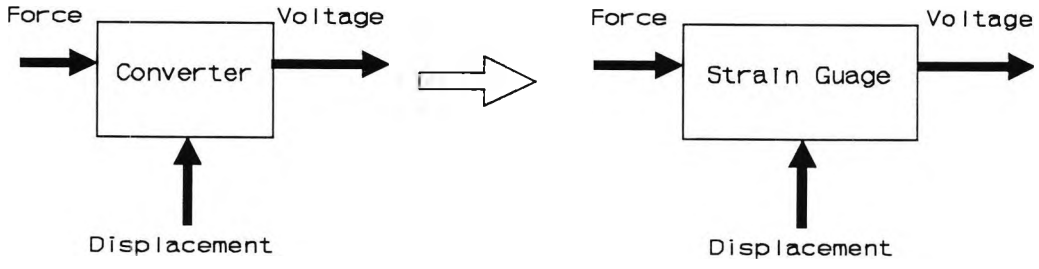
To aid design concept generation by systematic transformation of existing concepts the KBS uses a set of rules which generate functional configurations in the same PROLOG representation as that used for design concept generation by use of existing concepts. The rules require the following parameters: a description of the function to be transformed, lists of the input, output and control signals of the function, a list of intermediate signals which may be inserted in the resulting configuration, and the method of transformation to apply, i.e. replacement by subfunctions or by more abstract functions, or decomposition by energy conversion or transformation into parallel or sequential structures. They are listed below in a PROLOG-like notation with an accompanying graphical description for each rule and an example transformation generated by each rule. Rule one states that a function may be transformed to a subfunction with the same input, output and control signals:

Rule 1



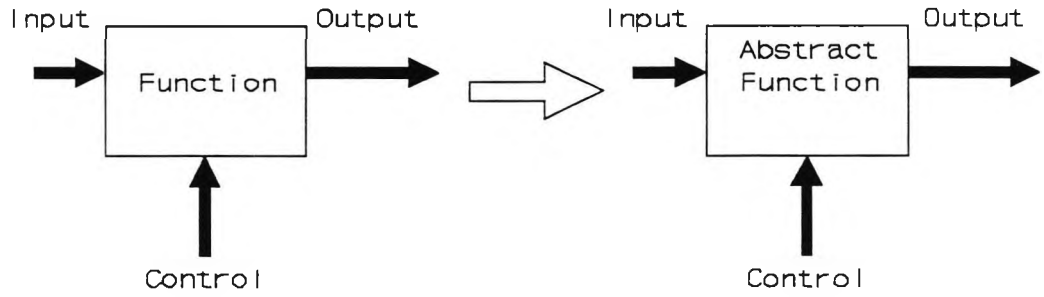

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           replace_by_subfunction,
           Result ) :-
                                    /* Find a subfunction in the knowledge
                                    /* base
      subfunction( Function, Subfunction,
                   Input_Signals, Output_Signals, Control_Signals ),
                                    /* Generate a number for the
                                    /* transformed function
      allocate_functions( [ Function_1 ] ),
                                    /* Define transformed function to be the
                                    /* subfunction with the same input,
                                    /* output and control signals
      Result = [ function( Function_1, Subfunction, _,
                           Input, Output, Control, _) ].
```

For example:



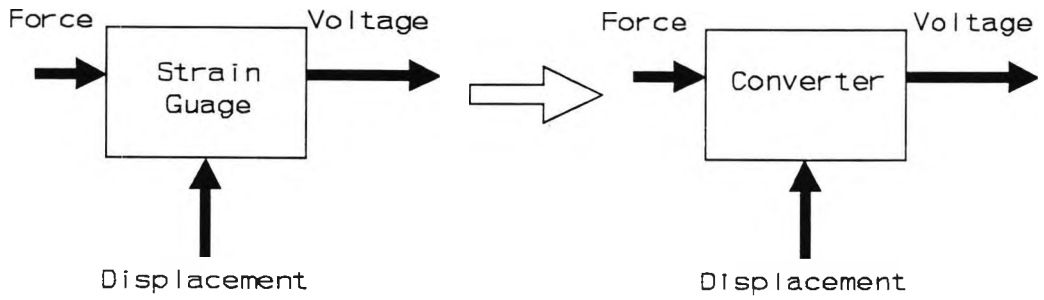

Rule two transforms a function to a more abstract function with the same input, output and control signals.

Rule 2




```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           replace_by_abstract_function,
           Result ) :-
                                        /* Find a more abstract function in the
                                        /* knowledge base
       abs_function( Function, Abstract_Function ),
                                        /* Generate a number for the
                                        /* transformed function
       allocate_functions( [ Function_1 ] ),
                                        /* Define transformed function to be the
                                        /* abstract function with the same input,
                                        /* output and control signals
       Result = [ function( Function_1, Abstract_Function, _,
                            Input, Output, Control, _) ].
```

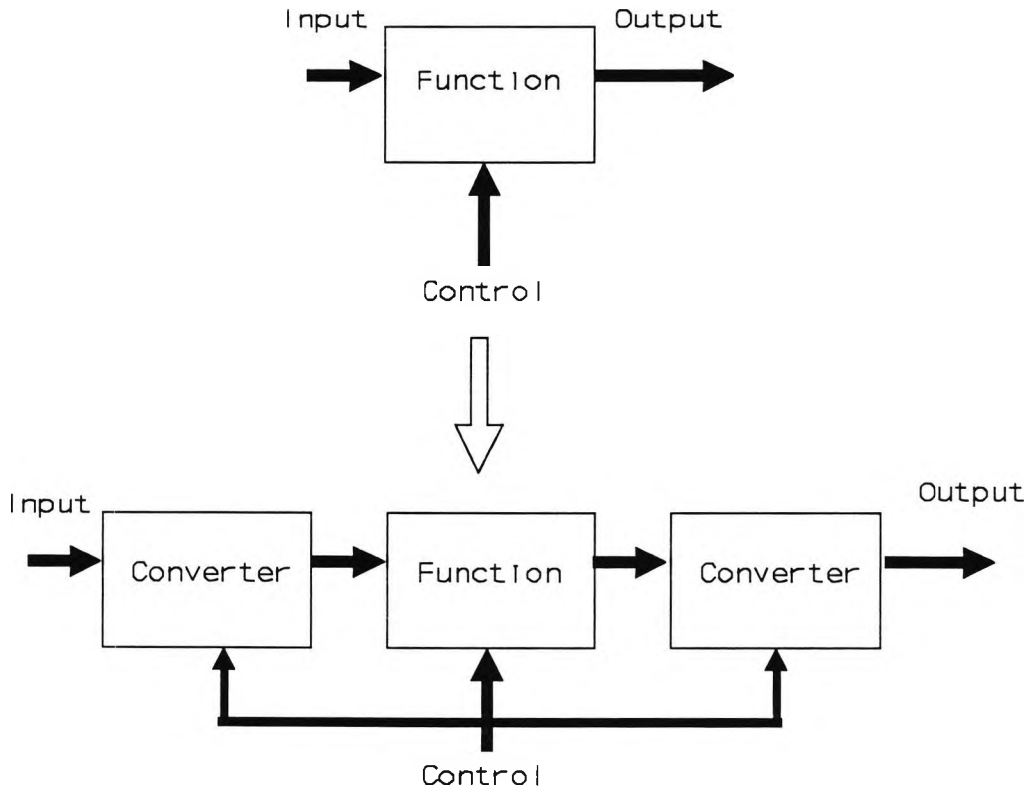
For Example:




Rules three, four and five are for functional decomposition by energy conversion; rule three states that a function may be decomposed by converting an input signal into an intermediate signal which then serves as input to the function and then converting the output from the function to the intended output; rule four

states that a control signal may be converted and the resulting signal applied as a control signal to the original function; rule five states that an intermediate signal may be output from the original function and then converted to the intended output signal.

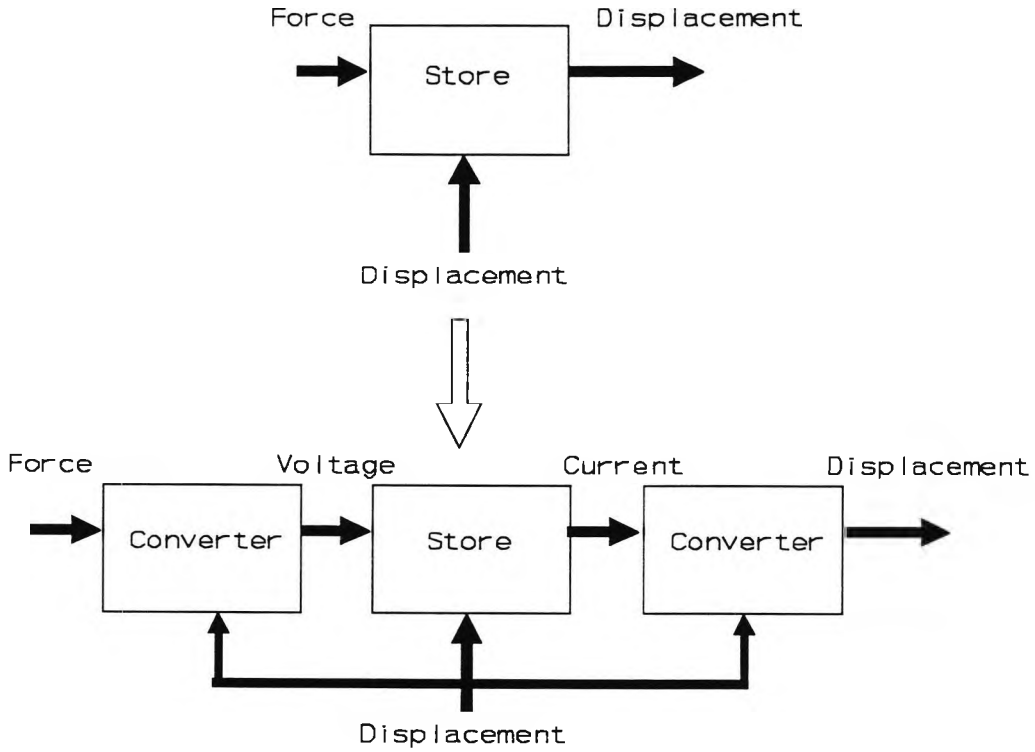
Rule 3



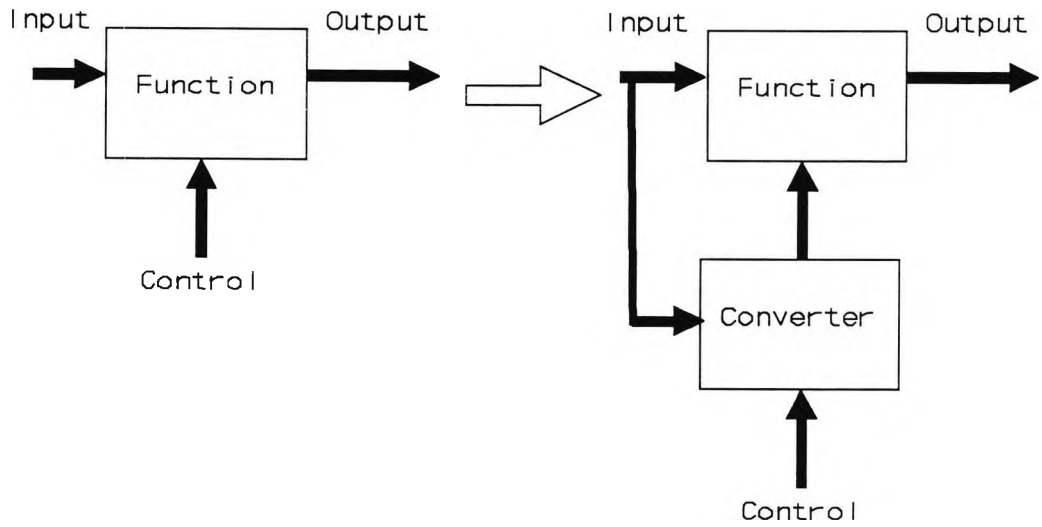

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signal, Output_Signal, Control_Signals,
           Intermediate_Signals,
           decompose_by_energy_conversion,
           Result ) :-
                                    /* Select an intermediate signal, store in
                                    /* Temp_1
      member( Temp_1, Intermediate_Signals ),
                                    /* Select another, store in Temp_2
      member( Temp_2, Intermediate_Signals ),
                                    /* Check functions in resulting
                                    /* decomposition are realisable based on
                                    /* information in the knowledge base
```

```
realisable( converter,
            Input_Signal,
            Temp_1,
            Control_Signals ),
realisable( Function_Type,
            Temp_1,
            Temp_2,
            Control_Signals ),
realisable( converter,
            Temp_2,
            Output_Signal,
            Control_signals ),
                                  /* Generate 4 numbers for functions in
                                  /* the decomposed structure
allocate_functions( [ Function_1, Function_2, Function_3, Function_4 ] ),
                                  /* Generate port numbers for
                                  /* intermediate signals
allocate_ports( Temp_1, Source_1, Dest_1 ),
allocate_ports( Temp_2, Source_2, Dest_2 ),
                                  /* Define resulting decomposition
Result = [ function( Function_1, Function_Type, _,
                     Input, Output, Control, _),
           composed( Function_1,
                       [ Function_2, Function_3, Function_4 ] ),
           function( Function_2, converter, _,
                     Input, Source_1, Control, _),
           function( Function_3, Function_Type, _,
                     Dest_1, Source_2, Control, _),
           function( Function_4, converter, _,
                     Dest_2, Output, Control, _),
           signal( Source_1, Temp_1, _),
           signal( Dest_1, Temp_1, _),
           signal( Source_2, Temp_2, _),
           signal( Dest_2, Temp_2, _),
           connected( Source_1, Dest_1, _),
           connected( Source_2, Dest_2, _) ].
```

For example:



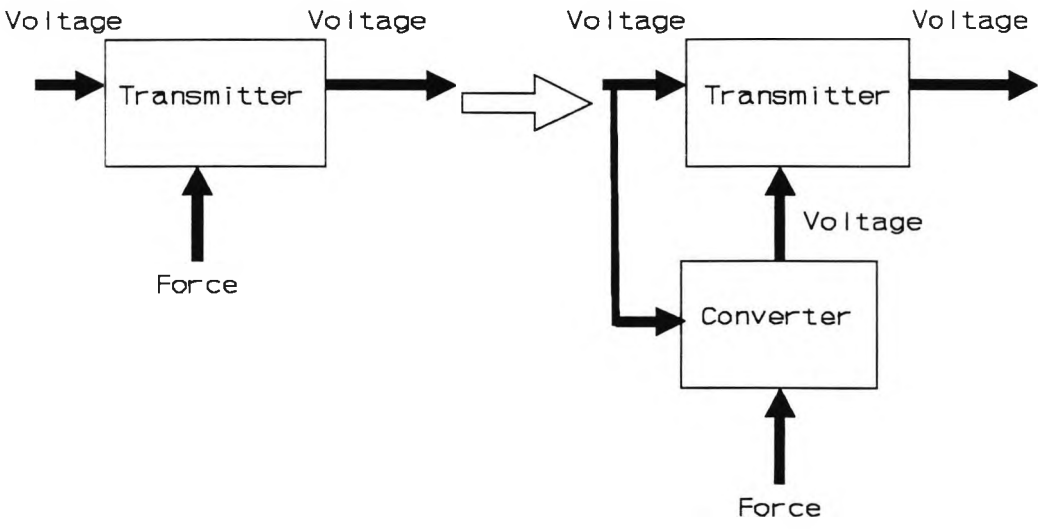

Rule 4



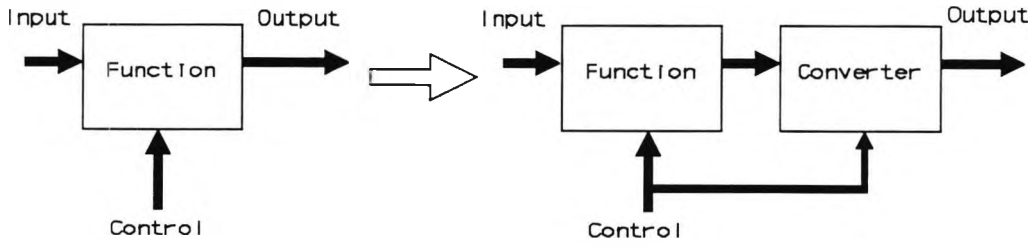

transform( function( Function_Number, Function_Type, Name,
Input, Output, Control, Coordinates ),
Input_Signals, Output_Signals, Control_Signal,
Intermediate_Signals,
decompose_by_energy_conversion.

```
        Result ) :-
                                        /* Select an intermediate signal, store in
                                        /* Temp
    member( Temp, Intermediate_Signals ),
                                        /* Check functions in resulting
                                        /* decomposition are realisable based on
                                        /* information in the knowledge base
    realisable( converter,
                Input_Signals,
                Temp,
                Control_Signal ),
    realisable( Function_Type,
                Input_Signals,
                Output_Signals,
                Temp ),
                                        /* Generate 3 numbers for functions in
                                        /* the decomposed structure
    allocate_functions( [ Function_1, Function_2, Function_3 ] ),
                                        /* Generate port numbers for
                                        /* intermediate signal
    allocate_ports( Temp, Source_Port, Dest_Port ),
                                        /* Define decomposition to be the
                                        /* same function but with the Control
                                        /* signal converted
    Result = [ function( Function_1, Function_Type, _,
                         Input, Output, Control, _),
               composed( Function_1, [ Function_2, Function_3 ] ),
               function( Function_2, converter, _,
                         Input, Source_Port, Control, _),
               function( Function_3, Function_Type, _,
                         Input, Output, Dest_Port, _),
               signal( Source_Port, Temp, _),
               signal( Dest_Port, Temp, _),
               connected( Source_Port, Dest_Port, _) ].
```

For example:



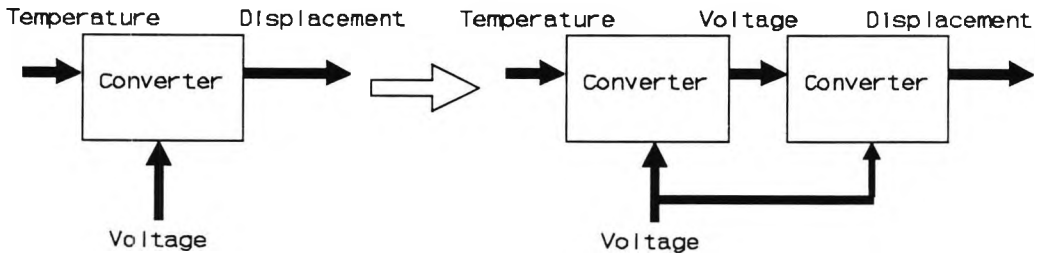

Rule 5




```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signal, Control_Signals,
           Intermediate_Signals,
           decompose_by_energy_conversion,
           Result ) :-
                                    /* Select an intermediate signal, store in
                                    /* Temp
      member( Temp, Intermediate_Signals ),
                                    /* Check functions in resulting
                                    /* decomposition are realisable based on
                                    /* information in the knowledge base
      realisable( Function_Type,
                  Input_Signals,
                  Temp,
                  Control_Signals ),
      realisable( converter,
                  Temp,
                  Output_Signal,
```

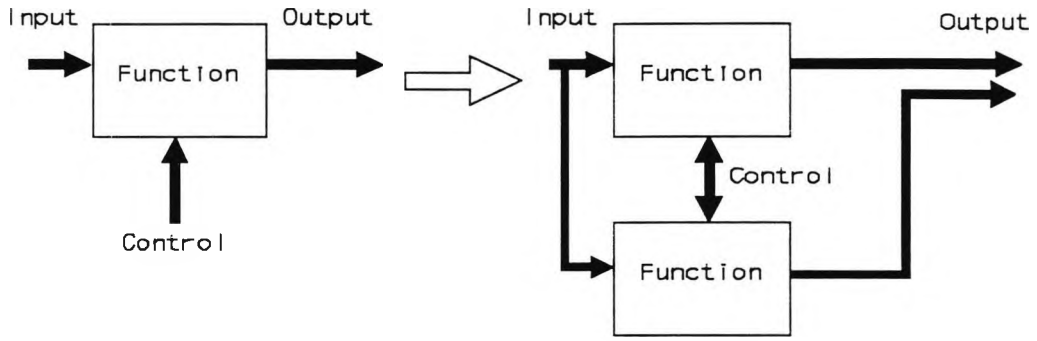
```
                Control_Signals ),
                                    /* Generate 3 numbers for functions in
                                    /* the decomposed structure
    allocate_functions( [ Function_1, Function_2, Function_3 ] ),
                                    /* Generate port numbers for
                                    /* intermediate signal
    allocate_ports( Temp, Source_Port, Dest_Port ),
                                    /* Define decomposition to be the
                                    /* same function but with the output
                                    /* signal converted
    Result = [ function( Function_1, Function_Type, _,
                         Input, Output, Control, _),
               composed( Function_1, [ Function_2, Function_3 ] ),
               function( Function_2, Function_Type, _,
                         Input, Source_Port, Control, _),
               function( Function_3, converter, _,
                         Dest_Port, Output, Control, _),
               signal( Source_Port, Temp, _),
               signal( Dest_Port, Temp, _),
               connected( Source_Port, Dest_Port, _) ].
```

For example:




Rules six, seven and eight are for functional decomposition into parallel structures: rule six states that a function with more than one output is equivalent to two functions of the same type, both with the same input, each having at least one of the original outputs and the combined output being the same as before; rule seven states that a function with more than one input is equivalent to the summed outputs of two functions of the same type, with the same output, each having at least one of the original inputs and the combined input being the same as before; rule eight is the same as rule seven but is for decomposition by splitting control signals.

Rule 6



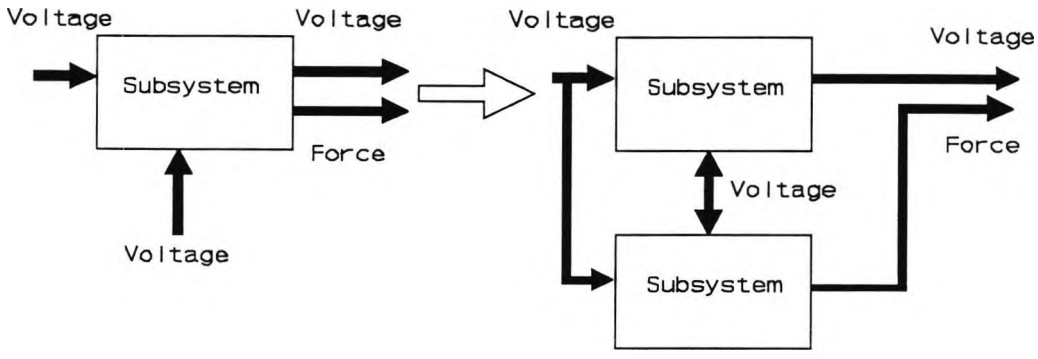

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           parallel_decomposition,
           Result ) :-
                                    /* Reorder list of output ports
      reorder( Output, Shuffled_Ports ),
                                    /* Split reordered list into 2 lists
      append( Output_1, Output_2, Shuffled_Ports ),
                                    /* 1st list contains at least 1 port
      Output_1 \== [],
                                    /* 2nd list contains at least 1 port
      Output_2 \== [],
                                    /* Find signals associated with the
                                    /* output port numbers
      signal( Output_1, Output_Signals_1, _),
      signal( Output_2, Output_Signals_2, _),
                                    /* Check functions in resulting
                                    /* decomposition are realisable based on
                                    /* information in the knowledge base
      realisable( Function_Type,
                  Input_Signals,
                  Output_Signals_1,
                  Control_Signals ),
      realisable( Function_Type,
                  Input_Signals,
                  Output_Signals_2,
                  Control_Signals ),
                                    /* Generate 3 numbers for functions in
                                    /* the decomposed structure
      allocate_functions( [ Function_1, Function_2, Function_3 ] ),
                                    /* Define decomposition
      Result = [ function( Function_1, Function_Type, _,
                           Input, Output, Control, _),
```

```
                composed( Function_1, [ Function_2, Function_3 ] ),
                function( Function_2, Function_Type, _,
                          Input, Output_1, Control, _),
                function( Function_3, Function_Type, _,
                          Input, Output_2, Control, _) ].
```

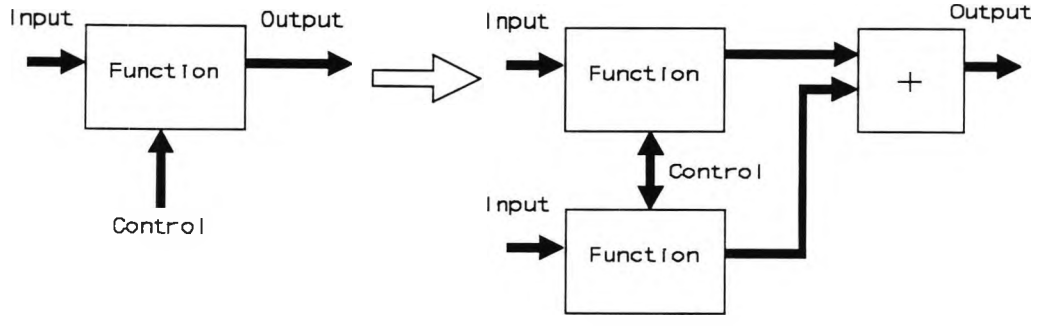
The reorder clause is defined as:

```
reorder( List, Shuffled_List ) :-
                                        /* Choose an item in the list
        member( X, List ),
                                        /* Remove it from the list
        delete( X, List, Temp_List ),
                                        /* Add it to the beginning of the list
        append( [ X ], Temp_List, Shuffled_List ).
```

For example:




Rule 7

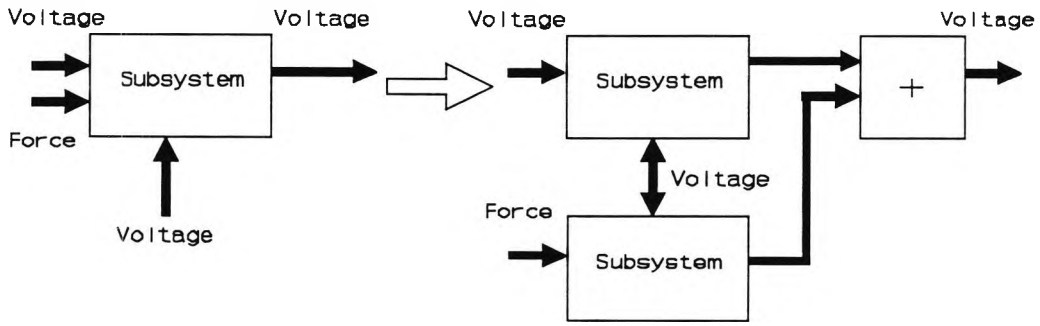



```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
```

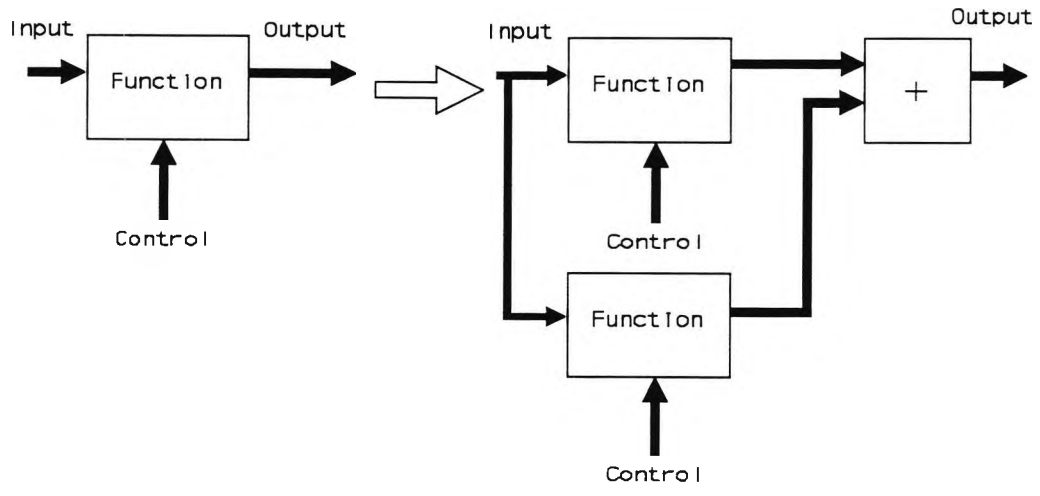
```
        Intermediate_Signals,
        parallel_decomposition,
        Result ) :-
                                    /* Reorder list of input ports
    reorder( Input, Shuffled_Ports ),
                                    /* Split reordered list into 2 lists
    append( Input_1, Input_2, Shuffled_Ports ),
                                    /* 1st list contains at least 1 port
    Input_1 \== [],
                                    /* 2nd list contains at least 1 port
    Input_2 \== [],
                                    /* Find signals associated with the input,
                                    /* port numbers
    signal( Input_1, Input_Signals_1, _),
    signal( Input_2, Input_Signals_2, _),
                                    /* Form a list of intermediate
                                    /* signals
    append( Output_Signals, Output_Signals, Signals ),
                                    /* Generate port numbers for
                                    /* intermediate signals
    allocate_ports( Signals, Function_Output, Sum_Input ),
                                    /* Split list of source ports into 2 lists
                                    /* of the same length
    append( Output_1, Output_2, Function_Output ),
    length( Output_1 ) == length( Output_2 ),
                                    /* Check functions in resulting
                                    /* decomposition are realisable based on
                                    /* information in the knowledge base
    realisable( Function_Type,
                Input_Signals_1,
                Output_Signals,
                Control_Signals ),
    realisable( Function_Type,
                Input_Signals_2,
                Output_Signals,
                Control_Signals ),
    realisable( summing_function,
                Signals,
                Output_Signals,
                [] ),
                                    /* Generate 4 numbers for functions in
                                    /* the decomposed structure
    allocate_functions( [ Function_1, Function_2, Function_3, Function_4 ] ),
                                    /* Define decomposition
    Result = [ function( Function_1, Function_Type, _,
                         Input, Output, Control, _),
               composed( Function_1,
```

```
                    [ Function_2, Function_3, Function_4 ] ),
    function( Function_2, Function_Type, _,
                    Input_1, Output_1, Control, _),
    function( Function_3, Function_Type, _,
                    Input_2, Output_2, Control, _),
    function( Function_4, summing_function, _,
                    Sum_Input, Output, [], _),
    signal( Sum_Input, Output_Signals, _),
    signal( Function_Output, Output_Signals, _),
    connected( Function_Output, Sum_Input, _) ].
```

For example:




Rule 8

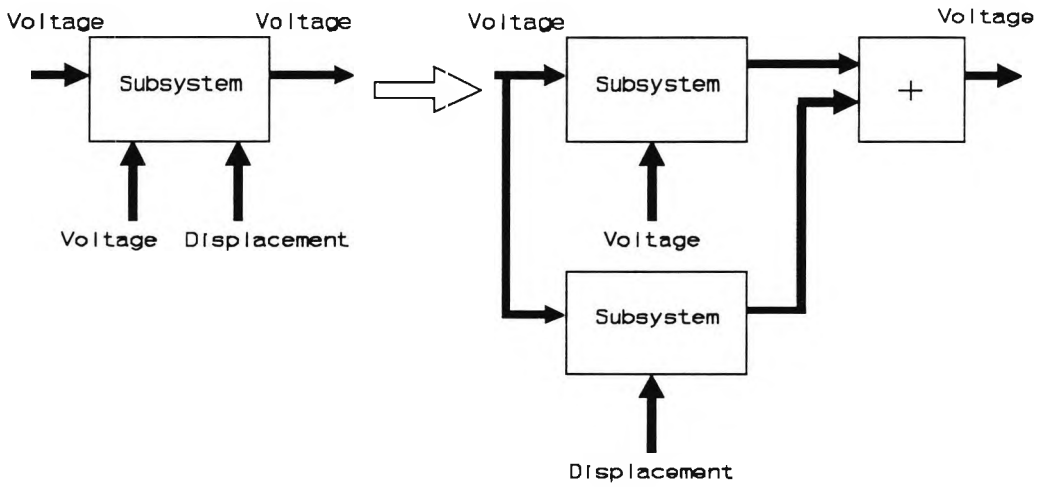



```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           parallel_decomposition,
           Result ) :-
```

```
                                        /* Reorder list of control ports
reorder( Control, Shuffled_Ports ),
                                        /* Split reordered list into 2 lists
append( Control_1, Control_2, Shuffled_Signals ),
                                        /* 1st list contains at least 1 port
Control_1 \== [],
                                        /* 2nd list contains at least 1 port
Control_2 \== [],
                                        /* Find signals associated with the
                                        /* control port numbers
signal( Control_1, Control_Signals_1, _),
signal( Control_2, Control_Signals_2, _),
                                        /* Form a list of intermediate
                                        /* signals
append( Output_Signals, Output_Signals, Signals ),
                                        /* Generate port numbers for
                                        /* intermediate signals
allocate_ports( Signals, Function_Output, Sum_Input ),
                                        /* Split list of source ports into 2 lists
                                        /* of the same length
append( Output_1, Output_2, Function_Output ),
length( Output_1 ) == length( Output_2 ),
                                        /* Check functions in resulting
                                        /* decomposition are realisable based on
                                        /* information in the knowledge base
realisable( Function_Type,
            Input_Signals,
            Output_Signals,
            Control_Signals_1 ),
realisable( Function_Type,
            Input_Signals,
            Output_Signals,
            Control_Signals_2 ),
realisable( summing_function,
            Signals,
            Output_Signals,
            [] ),
                                        /* Generate 4 numbers for functions in
                                        /* the decomposed structure
allocate_functions( [ Function_1, Function_2, Function_3, Function_4 ] ),
                                        /* Define decomposition
Result = [ function( Function_1, Function_Type, _,
                     Input, Output, Control, _),
           composed( Function_1,
                     [ Function_2, Function_3, Function_4 ] ),
           function( Function_2, Function_Type, _,
                     Input, Output_1, Control_1, _),
```

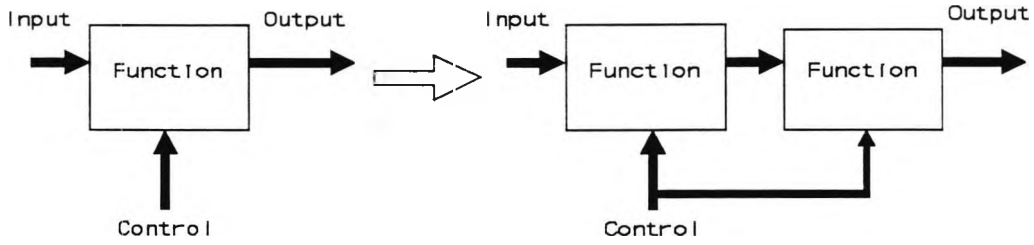
```
function( Function_3, Function_Type, _,
          Input, Output_2, Control_2, _),
function( Function_4, summing_function, _,
          Sum_Input, Output, [], _),
signal( Sum_Input, Output_Signals, _),
signal( Function_Output, Output_Signals, _),
connected( Function_Output, Sum_Input, _) ].
```

For example:



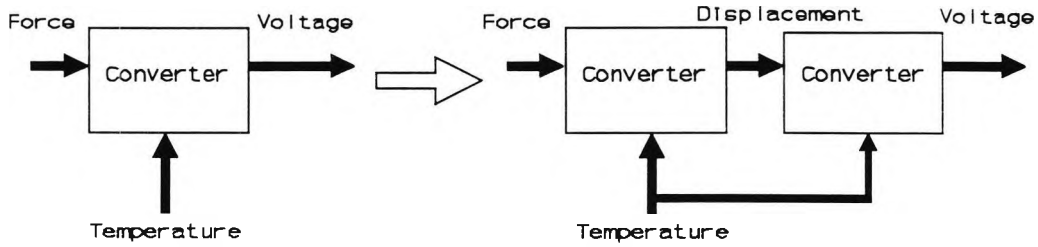

Rules nine and ten are for functional decomposition into sequential structures: rule nine states that a function may be decomposed into two identical functions with the output of one connected to the input of the other; rule ten states that the function may be decomposed into a sequence of connected functions of an arbitrary length.

Rule 9

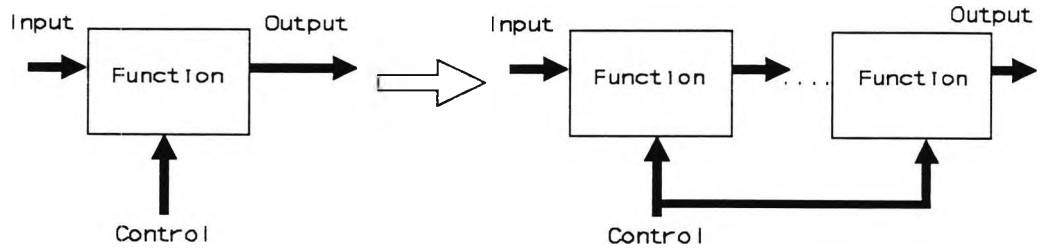



```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           sequential_decomposition,
           Result ) :-
                                    /* Select one or more intermediate signals
       append( Temp, _, Intermediate_Signals ),
       Temp \== [],
                                    /* Check functions in resulting
                                    /* decomposition are realisable based on
                                    /* information in the knowledge base
       realisable( Function_Type,
                   Input_Signals,
                   Temp,
                   Control_Signals ),
       realisable( Function_Type,
                   Temp,
                   Output_Signals,
                   Control_Signals ),
                                    /* Generate port numbers for the
                                    /* intermediate signals
       allocate_ports( Temp, Source_Ports, Dest_Ports ),
                                    /* Generate 2 numbers for functions in
                                    /* the decomposed structure
       allocate_functions( [ Function_1, Function_2 ] ),
                                    /* Define decomposition as two functions
                                    /* of the same type connected via the
                                    /* intermediate signals
       Result = [ function( Function_1, Function_Type, _,
                            Input, Source_Ports, Control, _),
                  function( Function_2, Function_Type, _,
                            Dest_Ports, Output, Control, _),
                  signal( Source_Ports, Temp, _),
                  signal( Dest_Ports, Temp, _),
                  connected( Source_Ports, Dest_Ports, _) ].
```

For example:



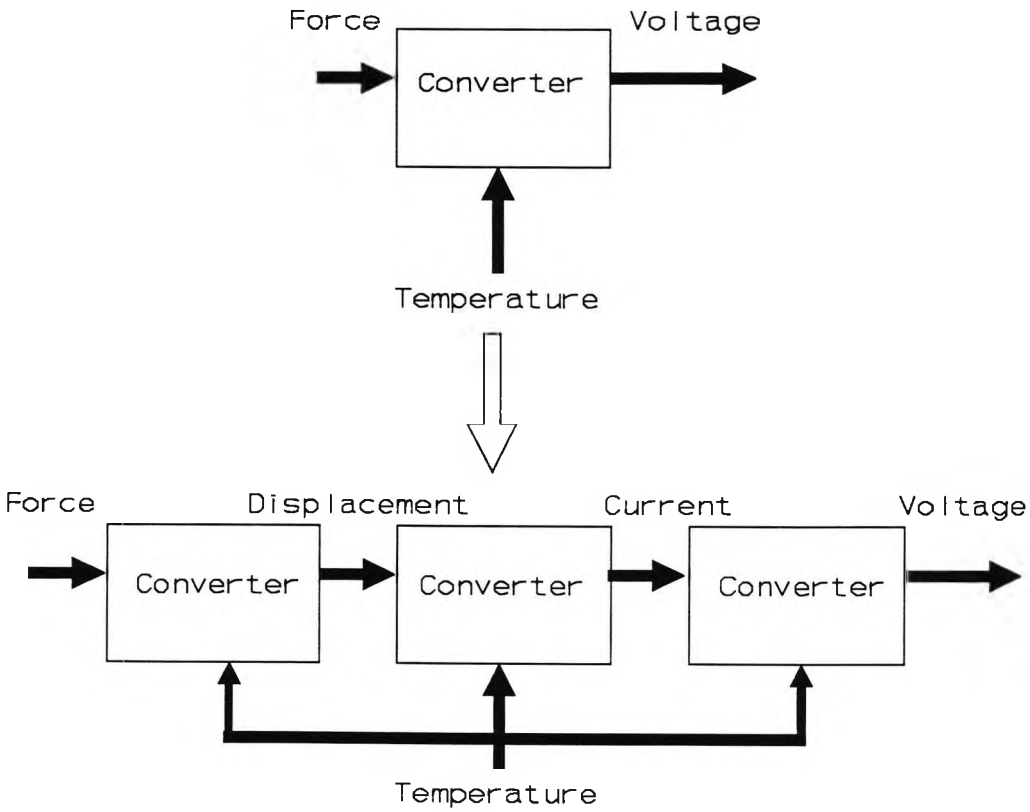

Rule 10




```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           sequential_decomposition,
           Result ) :-
                                   /* Select one or more intermediate signals
       append( Temp, _, Intermediate_Signals ),
       Temp \== [],
                                   /* Remove them from the list of
                                   /* intermediate signals
       delete( Temp, Intermediate_Signals, Updated_Intermediate_Signals ),
                                   /* Check functions in resulting
                                   /* decomposition are realisable based on
                                   /* information in the knowledge base
       realisable( Function_Type,
                   Input_Signals,
                   Temp,
                   Control_Signals ),
                                   /* Generate a number for a function in
                                   /* the decomposed structure
       allocate_functions( [ Function_1 ] ),
                                   /* Generate port numbers for the
                                   /* intermediate signals
```

```
allocate_ports( Temp, Source_Ports, Dest_Ports ),
                                    /* Find the decomposition of the
                                    /* function with the intermediate signals
                                    /* as input
transform( function( Function_Number, Function_Type, Name,
                     Dest_Ports, Output, Control, Coordinates ),
           Temp, Output_Signals, Control_Signals,
           Updated_Intermediate_Signals,
           sequential_decomposition,
           Partial_Result ),
                                    /* Define decomposition as a sequence
                                    /* of functions of the same type linked
                                    /* by the connecting signals
Result = [ function( Function_1, Function_Type, _,
                     Input, Source_Ports, Control, _),
           signal( Source_Ports, Temp, _),
           signal( Dest_Ports, Temp, _),
           connected( Source_Ports, Dest_Ports, _) |
           Partial _Result ].
```

For example:



The above rules all apply a single method of transformation, but they could be used for transformation by more than one method by invoking the rules sequentially. This could be done by the following additional rule which expects a list of transformation methods rather than a single method:

Rule 11

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Input_Signals, Output_Signals, Control_Signals,
           Intermediate_Signals,
           Methods,
           Result ) :-
                                    /* Select a method of transformation to
                                    /* apply
    member( Method, Methods ),
                                    /* Apply the method
    transform( function( Function_Number, Function_Type, Name,
                         Input, Output, Control, Coordinates ),
               Input_Signals, Output_Signals, Control_Signals,
               Intermediate_Signals,
               Method,
               Result ).
```

### 6.3.3 Support for Design Concept Generation by use of Analogy

To aid design concept generation by use of analogy the KBS uses a single rule which generates functional configurations by transforming analogous functional configurations of physical systems in the knowledge base of known functional configurations. The rule requires the following parameters: a description of the function to be transformed, and lists of analogous input, output, and control signals; the designer can also have the option of using any of the original signals as analogous signals if he does not wish to use analogy with those signals. The rule is shown below.

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Analogous_Input, Analogous_Output, Analogous_Control,
           transform_by_analogy,
           Result ) :-
                                    /* Find a functional configuration in the
                                    /* knowledge base with the analogous
                                    /* input, output and control signals
      find_config( Analogous_Input,
                   Analogous_Output,
                   Analogous_Control,
                   Analogous_Config ),
                                    /* Extract a list of the functions in the
                                    /* retrieved configuration
      find_functions( Analogous_Config,
                      Functions,
                      Rest ),
                                    /* Transform the list of functions their
                                    /* most abstract form
      abstraction( Functions, Abstract_Functions ),
                                    /* Replace extracted functions in the
                                    /* retrieved functional configuration with
                                    /* the abstracted functions
      append( Abstract_Functions, Rest, Abstract_Config ),
                                    /* Replace analogous signals in resulting
                                    /* functional configuration
      replace_signals( Abstract_Config,
                       Input, Analogous_Input,
                       Output, Analogous_Output,
                       Control, Analogous_Control,
                       Result ).
```

The action of the rule is as follows:

1) Find an analogous functional configuration in the knowledge base.

2) Transform the configuration to its most abstract form.

3) Replace analogous signals in the transformed configuration to give the resulting functional configuration.

The above rule is for analogy with systems having a particular set of analogous

input, output, or control signals. Analogy using more than one set of analogous signals is made possible by the following rule which expects a list of sets of analogous input, output, and control signals rather than a single set of signals:

```
transform( function( Function_Number, Function_Type, Name,
                     Input, Output, Control, Coordinates ),
           Analogous_Inputs, Analogous_Outputs, Analogous_Controls,
           transform_by_analogy,
           Result ) :-
                                    /* Select a set of analogous input signals
      member( Analogous_Input, Analogous_Inputs ),
                                   /* Select a set of analogous output signals
      member( Analogous_Output, Analogous_Outputs ),
                                    /* Select a set of analogous control
                                    /* signals
      member( Analogous_Control, Analogous_Control ),
                                    /* Transform the function by analogy
                                    /* using the selected signals
      transform( function( Function_Number, Function_Type, Name,
                           Input, Output, Control, Coordinates ),
                 Analogous_Input, Analogous_Output, Analogous_Control,
                 transform_by_analogy,
                 Result ).
```

### 6.3.4 Support for Design Concept Generation by use of First Principles

To aid design concept generation by use of first principles the KBS can generate sequences of physical laws, each of which forms a suggestion of the working principle of a realisation of a function. Each law in a sequence represents a realisable transition between physical signals and the output from one transition acts as the input for the next. The sequences are generated from the knowledge base of laws of physical effects. The following PROLOG-like rules illustrate how a sequence of laws which can form the working principle of a function can be generated; they could also be used to check if another signal can influence any which occur in a sequence of laws and if so generate the sequence of laws which describes the influence. The rules require as parameters the input, output and control signals of the function and a list of signals which may appear in a sequence.

```
generate_principle( Input, Output, Control, Intermediate_Signals, [ Law ] ) :-
                                        /* Find a law in the knowledge base for
                                        /* a transition from the input to the
                                        /* output with the control signal as a
                                        /* modulator
        generate_law( Input, Output, Control, Law ).


generate_principle( Input, Output, Control, Intermediate_Signals, Result ) :-
                                        /* Select an intermediate signal
        member( Signal, Intermediate_Signals ),
                                        /* Remove it from the list
        delete( Signal, Intermediate_Signals, Updated_Intermediate_Signals ),
                                        /* Find a law in the knowledge base for
                                        /* a transition from the input to the
                                        /* intermediate signal with the control
                                        /* signal as a modulator
        generate_law( Input, Signal, Control, Law ),
                                        /* Find a working principle for a
                                        /* transition from the intermediate signal
                                        /* to the output with the control signal
                                        /* as a modulator
        generate_principle( Signal, Output, Control,
                            Updated_Intermediate_Signals,
                            Partial_Result ),
                                        /* The sequence representing the final
                                        /* working principle is formed by adding
                                        /* the law to the beginning of the
                                        /* working principle for the transition
                                        /* from the intermediate signal to the
                                        /* output with the control signal as a
                                        /* modulator
        append( [ Law ], Partial_Result, Result ).
```

The first rule accesses a single law in the knowledge base which converts the input signal to the output signal using the control signal as a modulator, e.g. a function which converts light to voltage can be realised using the photovoltaic effect (see Table 5-2). The next rule generates a sequence of two or more laws, each law representing either a transition to one of the intermediate signals, or a transition to the output signal which is the last transition in the sequence, e.g. a function which converts light to voltage can be realised by a radiation heating effect in series with a thermoconductivity effect (see Table 5-2).

## 6.4 The Man-Machine Interface of the System

An interface to the KBS should allow the designer to perform the following actions, which when combined enable a graphical representation of a functional structure to be created:

- Add a specification for a function to the representation.

- Edit a specification for a function

- Delete one or more specifications of functions

- Move the graphical position of a specification of a function

- Connect the ports of two functions together

- Disconnect the connection between two ports

- Define the configuration of functions which a function is to be composed of at the next level of abstraction in the representation

- Move to a higher level of abstraction in the representation

- Save the current description

- Retrieve a previous description

- Move a portion of the representation to another part of it

- Copy a portion of the representation to another part of it

- Define a view

- Exit from the definition of a view

- Edit a view

- Copy one view to another

- Delete a view

The following options could aid the generation of solutions for a particular view:

- Examine the overall solution characteristics of all possible realisations of a function, taking into account any desired functional constraints

- Examine the solution characteristics of an individual component which realises a function and satisfies any desired functional constraints

- Invoke a design concept generation method to generate a functional structure

- Generate compatible solutions which meet the original specification and examine the characteristics of each component in the solution and the overall solution characteristics

- Save a representation of any generated solution configuration and the associated characteristics

- Calculate the overall solution characteristics for a configuration of functions at one level of abstraction using the scheme described in chapter four

- Verify that the functional constraints in a structure are valid at the different levels of abstraction in the structure using the scheme described in chapter four

- Check a functional structure for consistency, e.g. check the amplitude ranges

at connected ports are the same

- Verify a functional decomposition of a specification is consistent with the original specification

## 6.5 An Example of the Use of the KBS

What follows illustrates a session of possible interaction between a designer and the KBS. If a functional specification requires design concepts to be generated for a converter function with an input of force, an output of voltage and a conversion factor of between 0 to 100 V/N the interaction could proceed as follows:

- Start with two functional decompositions of the functional specification, one consisting of a force to displacement converter in series with a displacement to voltage converter, and one consisting of a force to current converter linked to a current to voltage converter.

- Experiment with constraining the conversion factors of each function in the decompositions and verifying the decompositions against the original functional specification. Select for further development the functional decompositions associated with each set of valid constraints.

- For each alternative configuration examine and store the overall solution characteristics of all realisations of each function.

- Explore the effects on these solution characteristics of adding other desired functional constraints and varying them. Store the results.

- For one or more of the alternative configurations explore the effects on the associated solution characteristics of altering the constraints on the conversion factors of each function and store the results; this can be done, for example by increasing the constraints, or by specifying a desired variation with a certain

signal, or by introducing a control signal into a configuration and specifying a desired variation with the control signal.

- Constrain one or more of the converter functions to be a subfunction and examine the effects on the associated solution spaces. Repeat this using different subfunctions and store all results.

- Invoke one or more design concept generation methods to generate functional configurations for each converter function and explore the resulting solution spaces. If appropriate, use a design concept generation method to generate further functional configurations, which would define in more detail the structure associated with any of the previously generated functions. Store all results.

- Constrain or expand the solution spaces stored by any method described in the previous steps. Store all results.

- Select one or more alternative configurations, generate compatible solutions to each and examine and store the resulting characteristics of these solutions.

- Select one or more of the previously generated solutions, add further desired constraints, and attempt to generate another compatible solution. Store any results.

- If necessary, repeat any of the above steps.

- Compare and contrast all results.

## 6.6 Summary

This chapter has illustrated the potential offered by automated support for design concept generation for instrument systems. The thesis concludes in the next chapter.

# CHAPTER 7

# CONCLUSIONS

## 7.1 Summary

A description has been given of a knowledge based system which aids the generation of design concepts for instrument systems. The KBS provides support facilities for design concept generation which should free up a designer to explore different types of solution and examine the effects of altering the solution space boundaries. These facilities are intended to stimulate the creativity of the designer and encourage many different types of solution to a problem to be considered. This has been illustrated by an example of the possible use of the system.

The design concepts generated must meet a functional specification which is defined in terms of a configuration of the functions performed by the system to be designed, and the associated constraints on each function; the set of constraints on each function define a specification of an energy transformation. Classifications have been given of the possible functions which may appear in a functional specification and the types of energy which may be transformed. The classification of functions is based on the storage, conversion, interconnection, control or supply of energy. The classification of energy forms is based on the energy flow which can be represented by a lumped parameter, distributed parameter, or ray approximation system model: a lumped parameter system model represents energy flow as a single quantity; a distributed parameter system model represents the variation of energy flow with space; a ray approximation system model represents energy flow which travels in

approximately straight lines and varies with direction. The functional specification may be represented at many levels of abstraction as the specification of any function can always be defined in further detail in terms of another configuration of functions and the associated constraints on each of them. A survey of automated aids for energy flow modelling has been carried out and a signal flow modelling scheme for the verification of a functional specification at each level has been proposed. This is based on finding the resulting specification of signal flow for each configuration of functions in a functional specification and checking it against the signal flow specification of the associated function.

Design concept generation by the following techniques is supported: use of existing concepts, systematic transformation of existing concepts, use of analogy, and use of first principles. The support offered by the system is based on a systematic search through the stored information in knowledge bases containing solution characteristics, known functional configurations, and laws of physical effects. Each knowledge base is represented by a PROLOG structure of frames. The knowledge base of solution characteristics contains information on the functions performed by an instrument system, sets of components which realise each function, and the functional characteristics of each component. The knowledge base of known functional configurations contains examples of configurations of functions which can be used to realise specific functions. The knowledge base of laws of physical effects contains equations for the laws of physics which can be used to form the working principle of a solution. Design concept generation by use of existing concepts is supported by the system enabling information retrieval from the knowledge base of known functional configurations. Systematic transformation of existing concepts is supported by a set of PROLOG rules, which when activated systematically transform a representation of a function. Use of analogy is supported by a PROLOG rule, which when activated retrieves an analogous functional configuration and transforms it using the analogy, which is that between physical signals. Use of first principles is supported by a set of PROLOG rules, which when activated generate a sequence of physical effects which can form the working principle of

a function.

## 7.2 Future Work

There are many avenues of further development for this research. The two main areas to consider are other design tasks to be supported and the expansion of the current system.

Other systems could be built to aid the analysis and evaluation tasks for all stages of design and the generation of candidate solutions for embodiment and detailed design; eventually they all might be integrated into one system. Another possibility is the implementation of a system for design development by a team of people. This would support the integration of different designs which have been developed separately and the generation of design concepts by group methods, e.g. by brainstorming.

The current KBS could be extended in many ways. Support facilities for the more creative methods of design concept generation could be introduced, e.g. the use of analogy with known functional configurations for other types of systems, particularly biological systems, could be investigated. Also, as the knowledge base of solution characteristics is bound to be incomplete it could be extended to store solution characteristics which are known to be not realisable and these would be used to save a designer from attempting to meet the specification of a function which cannot be realised. In addition, another possibility is the implementation of a blackboard system (Erman and Lesser, 1975; Englemore and Morgan, 1988) which would suggest alternative functional configurations when certain solution characteristics for a function are not stored. The classification of functions in the KBS could also be changed to include those functions for which an input signal does not influence the amplitude of an output signal, but influences another characteristic of the signal, e.g. the frequency or phase, as is the case with sensing using optical fibres. Configurations for energy conversion by optical fibre sensing could then be added to the system as well as information on optical fibre sensors, which would be classified in the way

described by Ning, Grattan, Yang and Palmer (1991). The use of the KBS could also be extended from instrument system design to control system design.

There are also many ways to extend the functional modelling scheme used by the KBS. The use of other modelling schemes needs to be investigated, such as power flow modelling, which would be the most accurate, and qualitative modelling (Forbus, 1984; Kuipers, 1986; Weld and de Kleer, 1990; Faltings and Strauss, 1992), which has the potential to be more efficient than signal flow modelling, but with a loss in accuracy. Each scheme could be used by a designer to verify a proposed design at a certain level of abstraction and in each scheme consideration could be given to the modelling of signal delays, digital signals, the effects of distance on transmission, the additional transformations performed by realisations of a function, e.g. the amount of energy storage associated with an energy conversion component, and the overall radiation pattern resulting from adding the individual radiation patterns in an array of ray approximation system functions, each separated by a certain distance.

# REFERENCES AND BIBLIOGRAPHY

**ABDULLLAH, F., FINKELSTEIN, L., KHAN, S.H. and HILL, W.J. (1993).**
Modelling in measurement and instrumentation.
*Proceedings of IMEKO TC1 and TC7 colloquium, City University, London*

**ADELI, H., (editor). (1988).**
Expert systems in construction and structural engineering. (London: Chapman and Hall).

**AKAGI, S. and FUJITA, K. (1987).**
Building an expert system for the preliminary design of ships.
*AI EDAM 1* (3): 191-205.

**BAILEY, W.N. and ABDULLAH, F. (1989).**
Interactive graphics-based computer-aided system dynamic modelling.
*Transactions of the Institute of Measurement and Control 11*: 127-137.

**BANNISTER, B.R. and WHITEHEAD, D.G. (1991).**
Instrumentation: transducers and interfacing. (London: Chapman and Hall).

**BARKER, H.A., CHEN, M. and TOWNSEND, P. (1988).**
Algorithms for transformations between block diagrams and signal flow graphs.
*Proceedings of the fourth IFAC Symposium on Computer Aided Design in Control Systems, Beijing.*

**BARKER, H.A., HARVEY, I.T. and TOWNSEND, P. (1993).**
Symbolic mathematics in control systems analysis and design.
*Transactions of the Institute of Measurement and Control 15*: 59-68.

**BARNEY, G.C. (1988).**
Intelligent instrumentation: microprocessor applications in measurement and control. (New York: Prentice-Hall).

**BARR, A. and FEIGENBAUM, E.A., (editors). (1981).**
The handbook of artificial intelligence Volume 1. (Los Altos, California: Morgan Kaufmann).

**BARR, A. and FEIGENBAUM, E.A., (editors). (1982).**
The handbook of artificial intelligence Volume 2. (Los Altos, California: Morgan Kaufmann).

**BARROW, H.G. (1984).**
VERIFY: a program for proving the correctness of digital hardware designs. *Artificial Intelligence 24*: 437-491.

**BENTLEY, J.P. (1988).**
Principles of measurement systems. (New York: Longman).

**BIRMINGHAM, W.P. and SIEWIOREK, D.P. (1988).**
Single board computer synthesis. In: Expert systems for engineering design: 113-139, edited by M.D. Rychener. (London: Academic Press).

**BOYD, R.W. (1983).**
Radiometry and the detection of optical radiation. (New York: John Wiley and Sons).

**BRACHMAN, R.J. and LEVESQUE, H.J., (editors). (1985).**
Readings in knowledge representation. (Los Altos, California: Morgan Kaufmann).

**BRADLEY, D.A., BRACEWELL, R.H. and CHAPLIN, R.V. (1993).**
Engineering design and mechatronics: the schemebuilder project. *Research in Engineering Design 4*: 241-248.

**BRADLEY, D.A., DAWSON, D., BURD, N.C. and LOADER, A.J. (1991).**
Mechatronics: electronics in products and processes. (London: Chapman and Hall).

**BRATKO, I. (1990).**
Prolog programming for artificial intelligence. (Wokingham: Addison-Wesley).

**BROOKES, A.M.P. (1968).**
Basic instrumentation for engineers. (London: Pergamon Press).

**BUCHANAN, B.G. and SHORTLIFFE, E.H., (editors). (1984).**
Rule-based expert systems. (Reading, Massachusetts: Addison-Wesley).

**BURTON, P.J. (1990).**
A model of the engineering design process. PhD thesis: The City University, School of Engineering.

**CLOCKSIN, W.F. and MELLISH, C.S. (1984).**
Programming in prolog. (New York: Springer-Verlag).

**COOK, S.C. (1990).**
A knowledge based system for computer-aided generation of measuring instrument specifications. PhD thesis: The City University, Department of Electrical, Electronic and Information Engineering.

**COX, B.J. (1986).**
Object-Oriented programming: an evolutionary approach. (Reading, Massachusetts: Addison-Wesley).

**CROSS, N. (1989).**
Engineering design methods. (New York: John Wiley and Sons).

**CUARADO, J.L. and CUARADO, C.Y. (1986).**
AI in computer vision.
*Byte January*: 237-258.

**DENYER, P.B. (1986).**
Silicon compilers. In: VLSI signal processing Volume 2: 3-13, edited by S-Y. Kung, R.E. Owen and J. Greg Nash. (New York: IEEE Press).

**DOEBELIN, E.O. (1972).**
System dynamics: modelling and response. (Columbus, Ohio: Merrill Publishing).

**DOEBELIN, E.O. (1985).**
Control system principles and design. (New York: John Wiley and Sons).

**DOEBELIN, E.O. (1990).**
Measurement systems: application and design. (New York: McGraw-Hill).

**ENGLEMORE, R. and MORGAN, T., (editors). (1988).**
Blackboard systems. (Wokingham: Addison-Wesley).

**ERMAN, L. and LESSER, V. (1975).**
A multi-level organisation for problem solving using many diverse cooperating sources of knowledge.
*Proceedings of the International Joint Conference on Artificial Intelligence*: 483-489.

**FALTINGS, B. and STRAUSS, P., (editors). (1992).**
Recent advances in qualitative physics. (Cambridge, Massachusetts: MIT Press).

**FINCHAM, W.H.A. and FREEMAN, M.H. (1974).**
Optics. (London: Butterworths).

**FINKELSTEIN, A.C.W., NUSEIBEH, B., FINKELSTEIN, L. and HUANG, J. (1992).**

Technology transfer: software engineering and engineering design.

*Computing and Control Engineering Journal November*: 259-265.

**FINKELSTEIN, L. (1977).**

Instrument science.

*Journal of Physics E: Scientific Instruments 10*: 566-572.

**FINKELSTEIN, L. (1992).**

State and trends of measurement science.

*Proceedings of the Australasian Instrumentation and Measurement Conference, Instrumentation and Measurement: Keys to Technological Advance, Centre for Continuing Education, University of Auckland, Auckland, New Zealand*: 1-6.

**FINKELSTEIN, L. and FINKELSTEIN, A.C.W. (1983).**

Review of design methodology.

*IEE Proceedings 130* (4), Part A: 213-222.

**FINKELSTEIN, L. and WATTS, R.D. (1978).**

Mathematical models of instrument systems - fundamental principles.

*Journal of Physics E: Scientific Instruments 11*: 841-855.

**FINKELSTEIN, L. and WATTS, R.D. (1983).**

Fundamentals of transducers: description by mathematical models. In: Handbook of measurement science Volume 2: 747-795, edited by P.H. Sydenham. (London: John Wiley and Sons).

**FORBUS, K. (1984).**

Qualitative process theory.

*Artificial Intelligence 24*: 85-168.

**FRENCH, M.J. (1985).**
Conceptual design for engineers. (London: Design Council).

**FROST, R.A. (1986).**
Introduction to knowledge base systems. (London: Collins).

**GERO, J.S., (editor). (1988).**
Artificial intelligence in engineering design Volumes 1 and 2. (Southampton: Elsevier).

**GERO, J.S., (editor). (1991).**
Artificial intelligence in design '91. (Oxford: Butterworth Heinemann).

**GRANT, P.W., JOBLING, C.P. and REZVANI, C. (1990).**
Some control applications of prolog.
*Proceedings of the Association for Logic Programming Conference, Bristol, UK.*

**HALLIDAY, D. and RESNICK, R. (1978).**
Physics Parts 1 and 2. (New York: John Wiley and Sons).

**HAYES-ROTH, F., WATERMAN, D.A. and LENAT, D.B., (editors). (1983).**
Building expert systems. (Reading: Addison-Wesley).

**HULTHAGE I., FARINACCI, M.L., FOX, M.S. and RYCHENER, M.D. (1988).**
Knowledge-Based alloy design. In: Expert systems for engineering design: 141-170, edited by M.D. Rychener. (London: Academic Press).

**HUNTER, A.B. (1991).**
Developments in artificial intelligence reasoning. In: Artificial intelligence in engineering: 295-335, edited by G. Winstanley. (Chichester: John Wiley and Sons).

**JACKSON, P. (1986).**
Introduction to expert systems. (Wokingham: Addison-Wesley).

**JENKINS, F.A. and WHITE, H.E. (1976).**
Fundamentals of optics. (New York: McGraw-Hill).

**JENKINS, T.E. (1987).**
Optical sensing techniques and signal processing. (Englewood Cliffs: Prentice-Hall).

**JOBLING, C.P. and GRANT, P.W. (1988).**
A rule-based program for signal flow graph reduction.
*Engineering Applications of Artificial Intelligence 1* (1): 22-33.

**JONES, J.C. (1980).**
Design Methods: seeds of human futures. (London: John Wiley and Sons).

**JONES, L.D. and FOSTER CHIN, A. (1991).**
Electronic instruments and measurements. (Englewood Cliffs: Prentice-Hall).

**KRAUSS, J.D. (1991).**
Electromagnetics. (New York: McGraw-Hill).

**KUIPERS, B. (1986).**
Qualitative simulation.
*Artificial Intelligence 29*: 289-338.

**LEIGH, J.R. (1992).**
Applied digital control. (Englewood Cliffs: Prentice-Hall).

**LEONARD, N.E. and LEVINE, W.S. (1992).**
Using MATLAB to analyse and design control systems. (Menlo Park, California: Benjamin Cummings).

**LIEBNER, R.D. (1981).**
Interactive functional modelling of instruments. PhD thesis: The City University, Department of Systems Science.

**LIN, T.Y. and STOTESBURY, S.D. (1981).**
Structural concepts and systems for architects and engineers. (New York: John Wiley and Sons).

**LOGAN, B., MILLINGTON, K. and SMITHERS, T. (1991).**
Being economical with the truth: assumption based context management in the Edinburgh Designer System. In: Artificial intelligence in design '91: 423-446, edited by J.S. Gero. (Oxford: Butterworth Heinemann).

**LUKAS, M.P. and DIXON, M.G. (1993).**
Knowledge based systems for conceptual design. Paper presented at the benefitting from knowledge based applications seminar at the Institute of Mechanical Engineers, London, January 1993.

**MAHER, M.L. (1988a).**
HI-RISE: An expert system for preliminary structural design. In: Expert systems for engineering design: 37-52, edited by M.D. Rychener. (London: Academic Press).

**MAHER, M.L. (1988b).**
Expert systems for structural design. In: Expert systems in engineering: 147-161, edited by D.T. Pham. (Bedford: IFS Publications).

**MAHER, M.L. and LONGINOS, P. (1987).**
Development of an expert system shell for engineering design.
*The International Journal of Applied Engineering Education.*

**MARION, J.B. (1981).**
Physics in the modern world. (New York: Academic Press).

**MAZDA, F.F. (1987).**
Electronic instruments and measurement techniques. (Cambridge: Cambridge University Press).

**McGILLEM, C.D. and COOPER, G.R. (1984).**
Continuous and discrete signal and system analysis. (New York: CBS Publishing).

**MICHIE, D., (editor). (1979).**
Expert systems in the microelectronics age. (Edinburgh: Edinburgh University Press).

**MIDDLEHOEK, S. and HOOGERWERF, A.C. (1986).**
Classifying solid state sensors: 'the sensor effect cube'.
*Sensors and Actuators 10*: 1-8.

**MIDDLEHOEK, S. and Van der SPIEGEL, J., (editors). (1987).**
Sensors and actuators: state of the art of sensor research and development. (Lausanne: Elsevier).

**MINSKY, M. (1975).**
A framework for representing knowledge. In: The psychology of computer vision: 211-277, edited by P.H. Winston. (New York: McGraw-Hill).

**MIRZA, M.K., NEVES, F.J.R. and FINKELSTEIN, L. (1990).**
A knowledge-based system for design-concept generation of instruments. *Measurement 8* (1): 7-11.

**NAGEL, L.W. (1975).**
SPICE2: A computer program to simulate semiconductor circuits. PhD thesis: University of California, Berkeley.

**NEUBERT, H.K.P. (1975).**
Instrument transducers. (Oxford: Clarendon Press).

**NING, Y.N., GRATTAN, K.T.V., WANG, W.M. and PALMER, A.W. (1991).**
A systematic classification and identification of optical fibre sensors. *Sensors and Actuators 29*: 21-36.

**PAHL, G. and BEITZ, W. (1988).**
Engineering Design: a systematic approach. (London: The Design Council).

**PHAM, D.T., (editor). (1988).**
Expert systems in engineering. (Bedford: IFS Publications).

**PUTTEN, van A.F.P. (1988).**
Electronic measurement systems. (New York: Prentice-Hall).

**QUILLIAN, M.R. (1968).**
Semantic memory. In: Semantic information processing: 227-270, edited by M. Minsky. (Cambridge, Massachusetts: MIT Press).

**RAMO, S., WHINNERY, J.R. and VAN DUZER, T. (1984).**
Fields and waves in communications electronics. (New York: John Wiley and Sons).

**RICH, E. and KNIGHT, K. (1991).**
Artificial intelligence. (New York: McGraw-Hill).

**RINGLAND, G.A. and DUCE, D.A., (editors). (1988).**
Approaches to knowledge representation: an introduction. (Taunton: Research Studies Press).

**ROBINSON, A.J. (1965).**
A machine-oriented logic based on the resolution principle.
*Journal of the ACM 12*: 23-41.

**ROOSENBURG, N.F.M., (editor). (1993).**
Proceedings of ICED 93. (Zurich: Heurista).

**RUPP, C.R. (1981).**
Components of a silicon compiler system. In: Proceedings VLSI 81 edited by J.P. Gray (London: Academic Press).

**SAADAT, H. (1993).**
Computational aids in control systems using MATLAB. (New York: McGraw-Hill).

**SEDGEWICK, R. (1983).**
Algorithms. (Reading: Addison-Wesley).

**SEIPPEL, R.G. (1983).**
Transducers, sensors and detectors. (Reston, Virginia: Reston Publishing Company).

**SRIRAM, D. and ADEY, R., (editors). (1986).**
Applications of artificial intelligence in engineering problems
Volumes 1 and 2. (Southampton: Computational Mechanics Publications).

**STEWARD, E.G. (1987).**
Fourier optics: an introduction. (Chichester: Ellis Horwood).

**SUBRAHMANYAM, P.A. (1986).**
Synapse: an expert system for VLSI design.
*Computer 19* (7): 78-89.

**SUH, N.P. (1990).**
The principles of design. (Oxford: Oxford University Press).

**TESKY, F.N. (1991).**
Representation and reasoning. In: Artificial intelligence in engineering: 33-63, edited by G. Winstanley. (Chichester: John Wiley and Sons).

**THE MATHWORKS INC. (1993).**
The MATLAB EXPO: an introduction to MATLAB, SIMULINK and the MATLAB application toolboxes.

**TILLEY, D.E. (1976).**
University physics for science and engineering. (Menlo Park, California: Cummings).

**VITKOVITCH, D., (editor). (1966).**
Field Analysis: experimental and computational methods. (London: Van Nostrand).

**WALTERS, J.R. and NIELSON, N.R. (1988).**
Crafting knowledge-based systems. (New York: John Wiley and Sons).

**WELD, D.S. and de KLEER, J., (editors). (1990).**
Readings in qualitative reasoning about physical systems. (San Mateo, California: Morgan Kaufmann).

**WELKOWITZ, W. and DEUTSCH, S. (1976).**

Biomedical instruments: theory and design. (New York: Academic Press).

**WELLSTEAD, P.E. (1979).**

Introduction to physical system modelling. (London: Academic Press).

**WILSON, J. and HAWKES, J.F.B. (1989).**

Optoelectronics: an introduction. (Englewood Cliffs: Prentice-Hall).

**WINSTON, P.H. (1984).**

Artificial intelligence. (Reading, Massachusetts: Addison-Wesley).

# APPENDIX 1

# AN OVERVIEW OF PROLOG

## A1.1 Data Definition in PROLOG

The PROLOG language is based on data objects called terms, of which there are the following types:

(i) *CONSTANTS*

A constant is an integer or real number, the possible values of which depends on the implementation, or an atom, which is a string of characters starting with a lower case character, e.g. tom, or enclosed in single quotes, e.g. 'Jerry'; certain special symbols such as {, }, +, * and [] are also defined as atoms.

(ii) *STRUCTURES*

A structure is a term composed of a functor, denoted by a name which must be an atom, and one or more arguments, each of which is another term, e.g. date( 1, january, 1994 ). The structure .(1, .(2, .(3, [] ))) is an example of a nested structure known as a list which is more usually written as a sequence of terms separated by commas and enclosed in square brackets, e.g. the above structure could be rewritten as [ 1, 2, 3 ]; [] is an atom known as the empty list. Lists are constructed or decomposed using the symbol '|' which separates or joins a head, which is a term, and a tail, which is another list, e.g. the above list could be represented as [ 1 | [ 2, 3 ]].

(iii) *VARIABLES*

A variable can represent the value of any unspecified term and is denoted by a

string of characters starting with a capital letter, e.g. Attribute, or underscore, e.g. _value. The variable denoted by a single underscore is known as the anonymous variable and can be used to represent several unrelated terms in the same expression when it is not necessary to know what the terms are but just that they exist, e.g. the above list could be represented as [ 1, _, _].

Two terms are said to 'match' if they are identical or they become identical after the variables in them are substituted by values which the variables are instantiated to, e.g. if the structures date( D, M, Y ) and date( 1, january, 1994) were matched this would result in variable D instantiated to 1, M instantiated to january and Y instantiated to 1994. In PROLOG this is written as:

date( D, M, Y ) = date( 1, january, 1994 )

and the output is:

D = 1, M = january, Y = 1994

The matching of two terms by the PROLOG interpreter is achieved using the principle of resolution (Robinson, 1965) which finds the most general match, i.e. the match with the least number of variable instantiations.

### A1.2 Programming in PROLOG

A PROLOG program consists of a database of clauses. A clause is a structure comprised of two terms, known as a head and a body, separated by the symbol ':-' which can be interpreted as 'if'. For example, the clause A :- B, C can be interpreted as goal A, which is the head, is true if goals B and C, which form the body, are true. Clauses can be divided into two types: those with a body which is always true and can be considered as facts, and those with one or more goals as a body and can therefore be considered as rules; each goal in the body of a rule is represented by another clause. A feature of PROLOG is the use of recursion in clauses, i.e. clauses that include themselves as a goal in the body of the rule they represent. The use of recursion is illustrated in the following example program which checks if a term is a member of a list, i.e. if it is present

in a list:

```
member( Term, [ Term | Tail ] ).

member( Term, [ Head | Tail ] ) :- member( Term, Tail ).
```

The first clause states that a term is a member of a list if it is at the head of the list. The second clause states that if a term is not at the head of a list it is still a member of the list if it is a member of the tail of the list.

A PROLOG program is run by posing a query consisting of a number of goals which the PROLOG interpreter attempts to solve by matching each goal with the heads of the clauses in the database, solving the goals in the body of the matched clauses and returning the answer 'yes' if this is successful or 'no' if it is not. For example, if the database contains the following facts and rule:

```
father( mary, george ).
father( john, george ).
father( sue, harry ).
father( george, edward ).

child( X, Y ) :- father( Y, X ).
```

the output from the system is 'yes' to the query child( george, mary ). Multiple solutions can be generated from the query child( X, Y ) which results in:

```
X = george, Y = mary;
X = george, Y = john;
X = harry, Y = sue;
X = edward, Y = george
```

where the semicolons are typed by the user and indicate a request for another solution.

# APPENDIX 2

# PUBLISHED WORK RESULTING FROM THIS RESEARCH

Measurement 11 (1993) 45–53
Elsevier



# Design-concept generation for instrument systems: A knowledge-based system approach

L. Finkelstein, R. Ginger, M. El-hami and M.K. Mirza

*Measurement and Instrumentation Centre, Department of Electrical, Electronic and Information Engineering, City University, London, UK*

**Abstract.** This paper presents further developments on a knowledge-based system (KBS) for conceptual design of instrument systems. The knowledge base of the KBS contains functional descriptions of physical components, commonly used in instrument systems. These consist of the input and output for a component, the operating signal, frequency ranges and impedance characteristics, the frequency response, the steady-state transfer function and the resolution for the component. The knowledge representation of the system is based on a hierarchical frame-based structure which has a highly modular and flexible nature. From the fundamental requirement specifications the KBS can generate a number of different combinations of physical subsystems which satisfy the requirements of an instrument designer at a power flow level of representation. An example to illustrate how the system works is finally presented.

**Keywords.** Conceptual design; Instrument system design; Knowledge-based system

## 1. Introduction

Computer-aided design (CAD) systems have over the years been shown to have a considerable and wide applicability in science and technology. CAD systems have been developed and used for drafting, modelling, numerical analysis etc, but have played little or no part in design-concept generation of instruments. However, over the past few years, development of computer aids based on artificial intelligence techniques for conceptual design of engineering systems has become an active area of research [1–3]. Mirza et al. have recently presented an account of a knowledge-based system for conceptual design of measurement instruments [4]. This paper reports the current state of developments.

Conceptual design of instrument systems can be viewed at a number of different levels of abstraction such as physical level, signal flow level, symbol flow level and

*Correspondence to:* Professor L. Finkelstein, Measurement and Instrumentation Centre, Department of Electrical, Electronic and Information Engineering, City University, Northampton Square, London EC1V 0HB, UK.

power flow level. A power flow level of abstraction describes the power flowing in and out of the instrument. Power can be described in terms of associated "effort" and "flow" variables, e.g. voltage and current, one of which is the signal carrying variable. Other levels of abstraction are possible, but the system implemented here is for the design of instruments from power flow level specifications. Conceptual design at a power flow level of representation involves finding the component or combination of components which can satisfy the fundamental requirements of an instrument system so that an acceptable level of signal is transferred from a source via the instrument to the receiver.

The fundamental requirement specifications are presented to the KBS in the form of the input and the output variables (e.g. force, voltage), the input signal range (the range of values over which the input signal will vary), the output signal range, the frequency range of the signal, the source impedance characteristic (a specification of how the source impedance varies with the input signal), the receiver impedance characteristic and the source frequency response (a specification of how the source impedance varies with frequency). From the designer requirements, the KBS then generates a number of viable design concepts. These are contrivances defined by some common principle of operations which could satisfy the design requirement. Problem decomposition and abstraction are considered and taken as the initial steps in generating design concepts. The system to be designed is decomposed into components, each of which is considered in terms of its function. Abstraction is essential in the process of identification of the component subfunctions into which the overall functional behaviour can be resolved. To generate design concepts for the elementary component functions a number of methods have been reviewed by Finkelstein and Finkelstein [5]. The KBS described here implements a convergent method for the purpose mentioned. The following sections outline the development of the KBS and its implementation.

## 2. The knowledge-based system (KBS)

The KBS consists of a knowledge base containing functional specifications related to power capability for components commonly used in measurement systems, frame handling utilities which store and retrieve information in the knowledge base and an inference engine which generates design concepts using the information in the knowledge base [6]. Figure 1 shows the overall flow of control in the KBS. The user interface provides a user-friendly interaction with

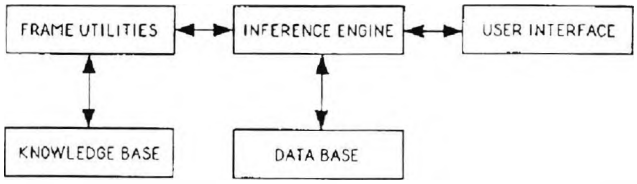



Fig. 1. The sequence of operations used by the knowledge-based system.

the system. Initially the designer specifies the requirements of an instrument to be designed. These consist of the input and output variables, the lower and upper bounds of the source and the receiver signal range, source and receiver impedance characteristics (series of impedance values associated with signal ranges) and the source frequency response which is specified in terms of a series of attenuation and frequency values. After the designer has specified the requirements, the KBS generates a solution. The designer is then free to ask for alternative solutions or specify a new design.

### *2.1. Knowledge representation*

The knowledge base is organised into a hierarchical frame-based structure. A frame is a data structure containing common information about a certain object or idea. Internally a frame contains a number of slots. Each slot contains a piece of information related to the overall information in the frame, e.g, the "input" slot of a frame named "beam" could contain the value "force". The knowledge base implemented here has slots called "effort" and "flow", "transfer" and "characteristic", which contain information on effort and flow variables, components which convert a specific input signal to a specific output signal and the characteristics for a particular component respectively. The classification of physical phenomena into effort and flow variables, which characterise power for lumped parameter systems, has been extensively studied [7–9]. The value in a slot can also refer to another frame and so define a relationship between the two frames. For instance, an "effort" slot could contain the value "voltage" for which a frame exists. In this way a structure of relationships between a number of frames can be built up. The function of the slots in the knowledge base is fully outlined in the following section.

### *2.2. The structure of the knowledge base*

In order to develop a compact and an efficient knowledge base, the instrument elements have been grouped together according to their common functional characteristics. The knowledge base developed here has a three-level hierarchy with the frames at each level having a common internal structure. This allows easy expansion to the knowledge base whenever necessary. An example of a typical part of the structure is shown in Fig. 2. The information represented at each level is as follows.

*Level 1*

There is only one frame at level 1. It contains a number of "effort" and "flow" slots, each containing information on a physical phenomenon which is an effort or flow and is itself represented by a frame at the next level down.

*Level 2*

At the second level each frame represents a physical phenomenon which is an effort or flow and is a signal-carrying variable. At present the frames contain a

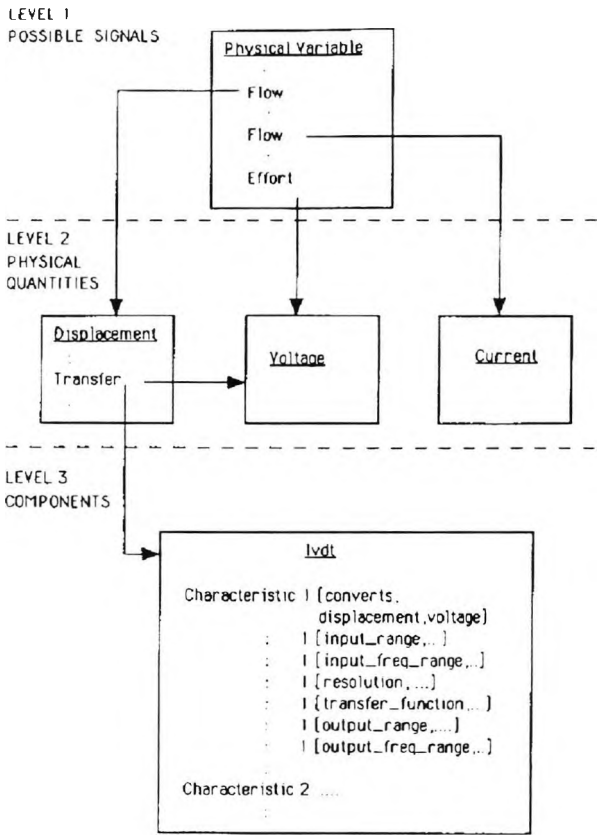



Fig. 2. Example portion of the knowledge base.

number of "transfer" slots. Each of these contains information on a component which would convert a physical phenomenon to another. The component and other physical phenomenon are also represented by frames, the former being at the next level down and the latter being at the current level. For example, in Fig. 2 the "transfer" slot of the "displacement" frame shows that an LVDT component can convert a displacement to a voltage.

*Level 3*

At the third level each frame represents a particular component and its characteristics. It may be necessary to store more than one set of characteristics for each particular component when the component is not a single-input single-output device and/or its characteristics vary with differing environmental conditions. Each frame has a number of "characteristic" slots containing information of the following form:

⟨set number⟩[⟨characteristic name⟩⟨data⟩]

Each set comprises the following:

- the input and output for the set;
- the input and output amplitude range;
- the input and output frequency range;
- the input and output impedance characteristic;
- the frequency response;
- the steady-state transfer function;
- the resolution.

It should be noted that different types of characteristic can easily be added if and when required. Some types of characteristic are represented by a linear piecewise approximation when this is appropriate. For instance, the steady-state transfer function is in the form of a list as follows:

[[⟨Input$_1$ Lower Bound⟩
⟨Input$_1$ Upper Bound⟩
⟨Output for Input$_1$ Lower Bound⟩
⟨Output for Input$_1$ Upper Bound⟩]
.
.
.
[⟨Input$_{n-1}$ Lower Bound⟩
⟨Input$_{n-1}$ Upper Bound⟩
⟨Output for Input$_{n-1}$ Lower Bound⟩
⟨Output for Input$_{n-1}$ Upper Bound⟩]]

### 3. Design concept generation

Conceptual design of an instrument can be viewed as a hierarchical planning process. Plans are produced at one level of detail and are then refined at the next level down until the lowest level is reached. The details involved in the process of generating design concepts using the KBS developed here are fully described in the following stages.

*Stage 1*

In the first stage, a functional description of the required transformation in terms of the signal transformation is produced. This has one of the following forms:

[function (⟨input variable⟩⟨output variable⟩) ]

or

[function (⟨input variable⟩⟨inter variable$_1$⟩)
function (⟨inter variable$_1$⟩⟨inter variable$_2$⟩)
.
.
.

function (⟨inter variable$_{n-1}$⟩⟨output variable⟩)]

A depth-first search is used, i.e. one in which all possible solutions are generated from the most recently generated partial solution first, before alternative solutions are generated from the next most recently generated partial solution. The search takes into consideration that a valid solution will not contain more than one occurrence of the same physical variable. The contents of the "effort" and "flow" slots in the "physical-variable" frame are used to generate the physical phenomena which will be the intermediate variables in the description.

*Stage 2*

During the second stage of the process, for each appropriate term produced by the first stage, the "transfer" slot is consulted to find the components that would convert the required input signal to the required output signal.

*Stage 3*

The third stage of the process begins by comparing the input and output frequency ranges of each component in the design concept with the required frequency range. If any do not lie within the required range another design concept will be generated. The components selected during stage 2 are then checked for signal range compatibility. First, the component to receive the input signal is checked to be operational within the input signal range. From this it can be determined if scaling and/or biasing signal elements are required. Next, a comparison of the resolution of the component with the signal range input to it is conducted. If the resolution is larger, signal amplification is needed. The output signal range of the component is then found from the appropriate steady-state transfer function and the input signal range. This is then used to check the signal range compatibility of the next component in cascade. This stage terminates by producing a modified list of components, compared with that of the previous stage, together with a list of signal ranges at each point.

*Stage 4*

In the final stage, impedance matching for signal preservation purposes between each pair of successive components is performed. The appropriate output and input impedance curves for successive components are looked up. For efforts the lowest and for flows the highest ratio of the input to output impedances within the signal range are found. If the ratios are unacceptable according to some predefined limit, extra matching elements are then introduced and linked in between the corresponding components.

The maximum attenuation over the operating frequency range of each component is also found from its frequency response. If this is unacceptably low, in order to obtain a level frequency response, a dynamic compensation component whose frequency response is the mirror image of that of the main component is inserted in cascade.

**4. Example**

This example illustrates a design-concept generated after presenting the fundamental requirement specifications to the KBS according to the previously indicated criteria. The system is required to generate design concepts for an instrument system to measure force with voltage as its output. The force has a range of between 0 and 5000 N, the voltage output varies between 10 and 10.1 V, the source impedance (effort/flow) is 100 kg $s^{-1}$ and the receiver impedance is 500 Ω for all values of voltage. An output from stage 1 of the design concept generation procedure could then be:

function (force, displacement),
function (displacement, voltage)

This clearly shows that a force-to-displacement conversion followed by a displacement-to-voltage conversion is one solution. Stage 2 could then produce the following output:

load-cell/force/displacement,
LVDT/displacement/voltage

This indicates that the required transformation can be realised using a load-cell connected to an LVDT. From this, stage 3 would then generate:

[0.0,5000.0] load-cell/force/displacement [0.0,0.001]
[0.0,0.001] LVTD/displacement/voltage [0.0,1.0]
[0.0,1.0] scaling component/10.0 [0.0,0.1]
[0.0,0.1] biasing component/+10.0 [10.0,10.1]

Stage 3 matches the signal ranges between components. The output shows the additional signal processing elements needed to do this and the signal ranges at each point. The numbers in the output represent constraints rather than absolute values. For instance, the term "[0.0,1.0] scaling component/10.0 [0.0,0.1]" means that the signal varying between 0 and 1.0 must be scaled down by a factor of at least 10.0 and so will then vary between 0 and 0.1 at most. The output from stage 4 would finally be:

match/100.0/10000.0 load-cell
match/load-cell-900.0/8.0-LVDT
match/LVDT-330.0/33000.0 scaling component
match/unknown/unknown biasing component
match/biasing component-5.0/500.0

Stage 4 matches the impedances between components. The output shows the highest and lowest impedances at each point. In this example, each match is acceptable and no extra impedance matching elements have been inserted. The term "match/100.0/10000.0 load-cell" means the load-cell has an input impedance (effort/flow) of at least 10000.0 kg $s^{-1}$, and this is connected to the output of the previous stage, the source in this case, which has a maximum output impedance

of 100 kg $s^{-1}$. It should be noted that, although the system can generate a number of different combinations if required, for simplicity only one configuration is generated and presented here.

## 5. Conclusions and future directions

A knowledge-based system (KBS) implementing a convergent method for design-concept generation of instrument systems has been successfully implemented. The KBS is capable of determining sets of alternative solution concepts after the establishment of the specification of the input/output requirements of the instrument system. The knowledge base contains functional descriptions of physical components commonly used in measurement systems. Due to the modular and flexible nature of the frame-based system, additional information can easily be incorporated into the knowledge base. The hierarchical frame-based structure of the knowledge base has been fully described. At present, the frames in the knowledge base contain "effort" and "flow", "transfer" and "characteristic" slots for a number of components used in instrument design. The software is written in PROLOG and runs on a Sun SPARC workstation and can easily be transferred to other machines supporting PROLOG e.g. an APPLE Macintosh.

The current KBS could further be extended for conceptual design of instrument control systems, multi-port instrument systems and evaluation of design concepts. Consideration of non-functional variables and environmental constraints for conceptual design is another area where more research is needed. Finally it is hoped and planned that, this work will be the precursor towards automated engineering design.

## Acknowledgements

The authors wish to thank the Science and Engineering Research Council (SERC) for financial support of this work and are grateful to the Measurement and Instrumentation Centre, City University for provision of facilities.

## References

[1] L. Finkelstein and A.C.W. Finkelstein. A review of instrument system design automation, in: G. Striker (ed.), *Acta IMEKO X, New Measurement Technology to Serve Mankind*, OIKK, Budapest, 1986.

[2] J.S. Gero, (ed.). *Artificial Intelligence in Engineering Design*, Elsevier/Computational Mechanics Publ., Amsterdam/Southampton, 1988.

[3] H. Yoshikawa, General design theory and CAD systems, in: T. Sata and E. Warman (eds.), *Man–Machine Communication in CAD/CAM*, North-Holland. Amsterdam, 1981.

[4] M.K. Mirza, F.J.R. Neves and L. Finkelstein. A knowledge-based system for design-concept generation of instruments, *Measurement* 8(1) (1990) 7–11.

# APPENDIX 2 - PUBLISHED WORK RESULTING FROM THIS RESEARCH



[5] L. Finkelstein and A.C.W. Finkelstein, Review of design methodology, *Proc. Inst. Electr. Eng. Part A* **130**(4) (1983) 213–222.
[6] J.L. Cuadrado and C.Y. Cuadrado, AI in computer vision, *Byte* (1986) 237–258.
[7] L. Finkelstein, Instrument science, *J. Phys. E Sci. Instrum.* **10** (1977) 566–572.
[8] L. Finkelstein and R.D. Watts, Fundamentals of transducers: Description by mathematical models, in: P.H. Sydenham (ed.), *Handbook of Measurement Science*, Vol. 2, 1983, pp. 747–795.
[9] L. Finkelstein and R.D. Watts, Mathematical models of instruments — Fundamental principles, *J. Phys. E Sci. Instrum.* **11** (1987) 841–855.