



## Contributed Paper

# Object-Oriented CAD for Practical Hydraulic System Design

P. K. WONG

The Hong Kong Polytechnic University, Hong Kong

T. P. LEUNG

The Hong Kong Polytechnic University, Hong Kong

C. W. CHUEN

The Hong Kong Polytechnic University, Hong Kong

(Received October 1995; in revised form May 1996)

---

*This paper describes a new approach to the design and implementation of object-oriented CAD for hydraulic systems. This CAD tool is an intelligent computer assistant for hydraulic engineers, to aid them in designing their own systems. Work has been devoted to a new class-modelling method for the object model of this CAD system. The method can model the behaviour of existing hydraulic circuits and components more rigorously, to avoid errors. Physical data about hydraulic systems has also been captured in the data model for another application, namely fluid power-system assembly problems, and this will be briefly described. An application example, a deep-drawing press, has been selected to illustrate the usefulness of the CAD system.*

Copyright © 1996 Elsevier Science Ltd

---

Keywords: Object-oriented models, intelligent computer-aided design (ICAD) system, hydraulic circuitry design.

## 1. INTRODUCTION

The expert-system approach is currently being widely applied to computer-aided design (CAD) for simple hydraulic systems.<sup>1-5</sup> Most of the expert systems used in the design of hydraulic systems are built from either rule-based or frame-based representations.<sup>2</sup> The main disadvantage of rule-based formalisms is the difficulty of representing exceptional information; hence, the knowledge-base becomes sizeable in further expansion and the overall system's behaviour is difficult to predict as the rules are so structurally independent. The main disadvantage of frame-based representations is the difficulty of defining causal relationships<sup>4</sup> or interactions between objects, as each object is passive and is normally actuated by inference engines. Those frames are not able to actuate other objects. In this paper, the construction of a new CAD approach for complex

hydraulic circuit design based on object-oriented technology is presented.

## 2. DEVELOPMENT OF A NEW CAD APPROACH FOR HYDRAULIC DESIGN PROBLEMS

A hydraulic circuit design process normally involves sketching a possible circuit layout and selecting appropriate hydraulic components. The basic functions of a hydraulic system are determined by circuit configurations, whereas the circuit's performance mainly relies on the sizes and types of the hydraulic components. In practice, hydraulic system design does not build from scratch; many designers like to tackle the problem by modularisation of the system, and then design each circuit module or sub-system by referring to many already existing, effective designs. Usually, such effective existing circuits and sub-systems can readily be adopted after careful adjustment of the sub-system components. The methods of adjustment and integration are determined on the basis of the designer's experience. Every designed circuit is only theoretically sound; fine-tuning of the hydraulic components is necessary for testing and commissioning after the real system is built.

---

Correspondence should be sent to: Professor T. P. Leung, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: vptleung@polyu.edu.hk.

This approach can be accomplished more efficiently in a computer with the aid of an effective database and its management system. However, if all the hydraulic systems in industry were required to be modelled in the conventional database format, the amount of information to be specified and stored would become unmanageably large, due to the combination and variation of hydraulic sub-systems and components. In addition to that, engineering data have a large variety of data types, with complex relationships among them. The concept of object-oriented representation has been introduced in the subject of artificial intelligence.<sup>6</sup> There are many papers discussing the idea of object-oriented knowledge bases, applied to different design domains.<sup>7-10</sup> Zhang<sup>7</sup> proposed using an object-oriented knowledge-base application to hydraulic design. He only defined the parameters of some standard hydraulic components in an object-oriented hierarchy. Each defined object is a set of static data that is triggered by production rules, and this is a design-from-scratch approach. Hence many design rules are still required. This is not an efficient approach to complicated design problems.

The recent development of object-oriented technology has led to a new approach to hydraulic design. Of particular importance is that object-oriented databases (OODB) provide a powerful knowledge-representation capability, as well as complex data definition. The new philosophy adopted in this paper is to represent various hydraulic circuits in the solution space by decomposing them into hydraulic sub-circuits and standard components, which are then represented and stored in an object-oriented database format that acts as an active design library.

In the proposed framework, design specifications are separated into functional requirements, performance goals, and constraints. Starting with functional requirements, a skeletal design comprising essential functions is retrieved from an object-oriented database. The functions in the skeletal scheme are then mapped to appropriate objects that either represent hydraulic components, or satisfy performance goals and constraints. The process of mapping functions (usually referring to attribute values/codes and the existence of functions) to hydraulic components/sub-circuits is directly related to the stringency of performance goals and constraints. The solution is formed by re-synthesising the retrieved sub-circuits and devices. Adjustment can be performed by referring to some adjustment sub-circuits or devices, and to the interactions of each sub-circuit. The proposed design approach, built with an object-oriented representation, is able to provide the following capabilities: (i) a high degree of modularity; flexible model building and easy modification; (ii) reduced information redundancy by means of the technique of inheritance, and hence a compact design library; (iii) data independence and security, by using the concept of encapsulation; (iv) shorter data-access times, due to the hierarchical structure of the object model; and (v) an infrastructure for supporting other fluid power system design activities.

### 3. OBJECT-ORIENTED MODELLING FOR HYDRAULIC SYSTEMS

To build an object-oriented data model for representing the design solutions, the development process can borrow from some of the ideas of current object-oriented software-development methodologies. At present, the methods commonly used in object-oriented software development are based on entity-relationship models such as those of Coad and Yourdon,<sup>11</sup> Booch,<sup>12</sup> or the object modelling technique (OMT),<sup>13</sup> but no single methodology has been found to be complete.<sup>14</sup> No matter what these methods are, they start from the identification of objects and classes from the problem domain, then the identification of the semantics and the relationships between these classes and objects, and finally the implementation. The development process for hydraulic applications based on the design algorithm proposed here may be broken down into the following iterative steps:

- (1) Identify all the possible and effective circuits in the problem domains at a circuit level of abstraction.
- (2) Decompose the circuit into modules/sub-systems.
- (3) Identify the properties of these sub-systems, and group them in accordance with the common properties, into classes.
- (4) Identify the component objects from the identified modules/sub-systems, and repeat Step 3 at a hydraulic component level of abstraction.
- (5) Identify the relationships among the classes and the objects.
- (6) Design the classes and the objects and the object-oriented data model.
- (7) Specify the interface and the implementation of the classes and the objects.

#### 3.1. Analysis and identification of hydraulic circuits

Generally speaking, the hydraulic domain is less modular than electronic circuits, so it is difficult to find some existing sub-circuits that satisfy the requirements of interchangeability and reusability. Besides, if a circuit is decomposed by the conventional single-function decomposition method, there will be many redundancies and contradictions in the configuration. In many real cases, a speed control sub-circuit integrated with a directional control sub-circuit will frequently not result in a circuit that can have both speed and directional control functions. With reference to many hydraulic circuits, a proper sub-circuit usually possesses two or more functions, for instance a regenerative circuit has at least a directional control function plus a speed control function. Besides, hydraulic subsystems can also be easily identified using the concept of a dynamic module, instead of finding a fixed sub-module. The dynamic module concept suggests that a sub-module can be decomposed into a network of a few functional blocks with conditional connections. The principles of object-oriented programming fully support the concept of dynamic modules; therefore, they can be represented as objects.

By an extensive analysis of many practical hydraulic circuits, plus abundant hydraulic design experience, a new decomposition concept of a general hydraulic circuit (Fig. 1) is proposed. A hydraulic circuit can be viewed as the building of primary function circuits (such as regenerative, pre-fill and synchronise circuits), secondary/adjustment function circuits (such as decompression and pressure reducing circuits) and power supply circuits (such as low-high pressure control power supply, constant power supply). Various hydraulic applications define different primary functions and secondary or adjustment functions as their default functions. The retrieval of sub-circuits for secondary functions depends on the user supply messages driven by the calculated parameters of the circuits for elementary functions (usually flow rate, pressure and cost factors are the main parameters for determining the types of circuits and components involved) and the interactions between the circuits.

### 3.2. Modelling of existing effective hydraulic subsystems

Representing a sub-system is not just using a symbol; there are many factors requiring consideration. Existing subsystems are all modelled as composite objects, encapsulating design specifications and organised in an object-oriented hierarchical structure for supporting property inheritance. Attributes of an existing sub-circuit  $S_{att}$  can be represented as:

$$S_{att} = \langle C, I, F, A, Conn, Inter \rangle.$$

Here  $C$  stands for the set of generic components already

available in the design library. Each sub-circuit only provides suitable pointers to the classes of the predefined components; the details of the components are determined by the component classes themselves.  $I$  is the interface of the circuit. It consists of the external input/output number, characteristics, and positions of the circuit; an example of the port positions is illustrated in Fig. 2(a).

$F$  characterises the functionalities and the nominal operating conditions of the design, which are the basic indexing attributes for selecting a sub-circuit, and ensure that the selected sub-circuit will not be beyond the operating limits. In an implementation, the attributes may be coded in forms of integers, floating points or strings.

$A$  denotes the set of performance attributes of the circuit characteristics; it usually refers to the methods and formula for calculating the sub-system output parameters. Production rules and mathematical functions are good implementation tools for this kind of attribute.

$Conn$  denotes the connectivity information (netlist description) of the internal components, in the form of text. The netlist file format is often used in the current CAD tools for designing electronic printed-circuit boards.

$Inter$  consists of various descriptions of the design constraints and interactions with other components/sub-circuits. The aim is to ensure compatibility between modules, and to eliminate illegal combinations. The interaction checking process mainly relies on checking the system parameters, class relationship and class names. For implementing the contradiction relationship among classes, the names of the contradicted classes are encapsulated in the corresponding modules. Every time a suitable module is

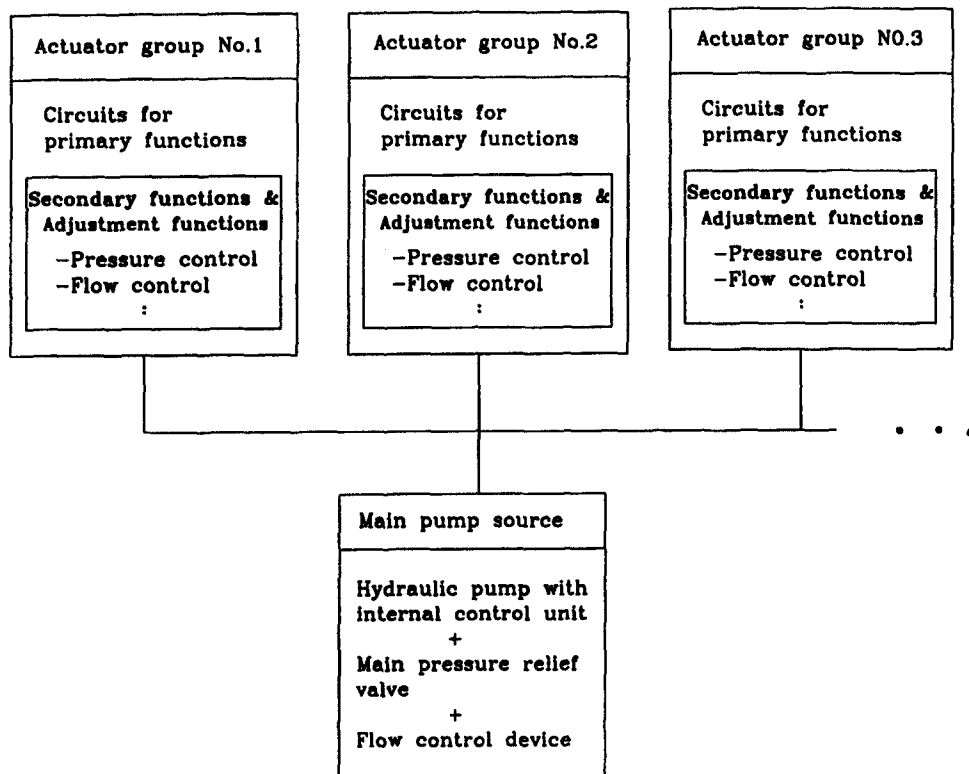


Fig. 1. Synthesis of hydraulic circuits.

retrieved, it will actuate the internal switches of its contradicted classes by "message passing".

An example of the key properties of the generic types of regenerative circuits is shown in Table 1 and Fig. 2. Figure 3 shows the data model of these regenerative circuits, which is constructed on the basis of the characteristics of object-

oriented technology, where the models are separated into sub-system and component layers. The techniques used for building this model are discussed in Section 3.4.

### 3.3. Modelling of standard hydraulic components

The standard hydraulic components are also organised in

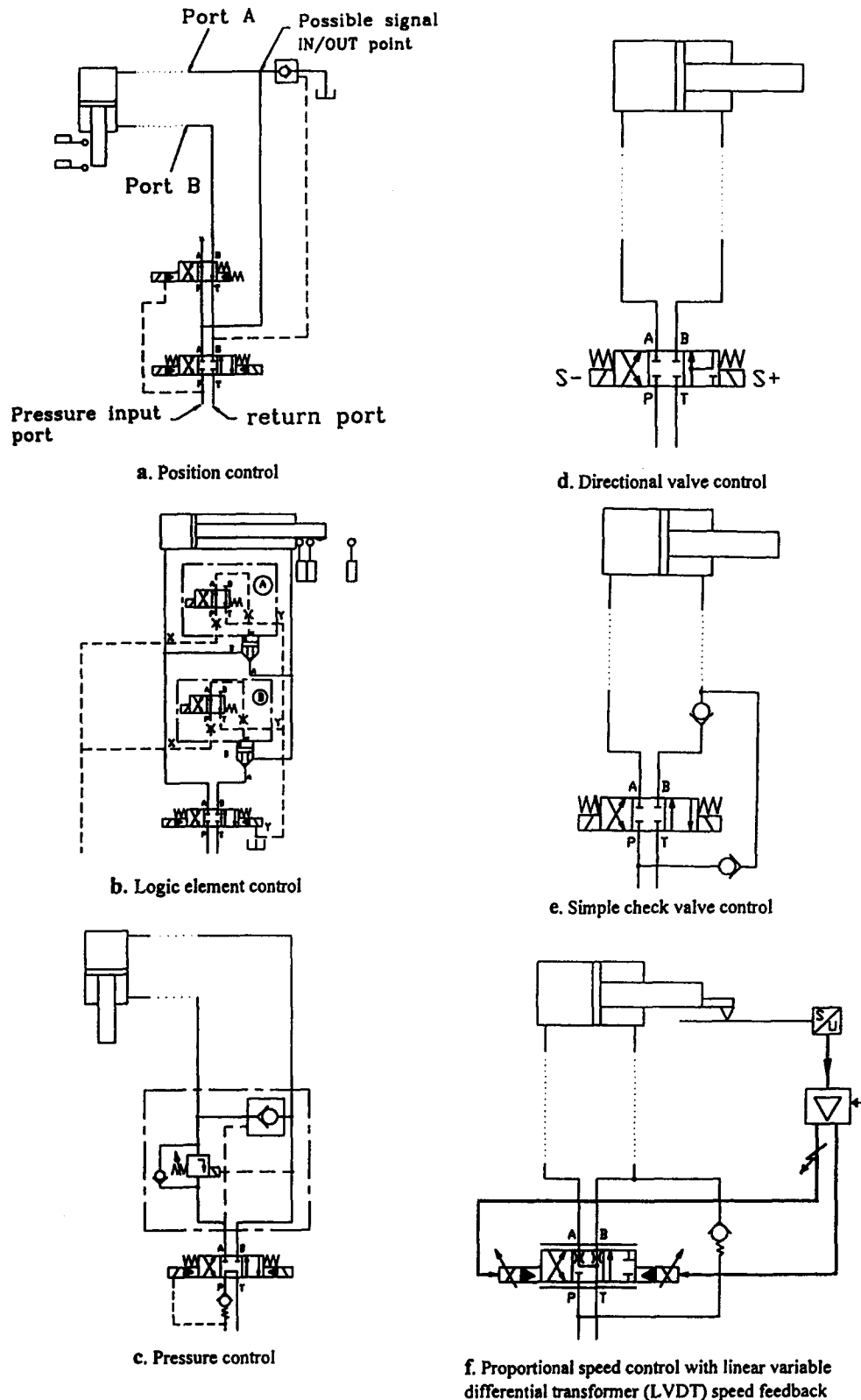
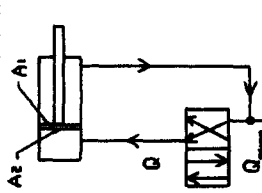


Fig. 2. Example of a generic class of existing effective hydraulic regenerative circuits.

Table 1. Key properties of the class of regenerative circuits

F	Basic selection criteria of regenerative circuits	<ul style="list-style-type: none"><li>90mm ≤ calculated bore diameter of the cylinder <b>D</b> &lt; 650 mm    AND</li><li>2 &lt; Speed ratio of high load to no load condition <math>\epsilon \leq 7</math>    AND</li><li>Stroke of actuator &gt; 300 mm</li></ul> <p>Remarks: A regenerative sub-circuit is one type of energy-saving speed and directional control. It is used for accelerating the speed of the piston extension. Under normal circumstances, the pump found in the database cannot supply oil at such a high flow rate.</p>						
Sub-class name	Types of regenerative circuits	Position control	Logic element control	Pressure control	Direction valve control	Simple check control	Proportional speed control	
C	Elements involved	Single-end-rod double acting cylinder						
		<ul style="list-style-type: none"><li>Limit switches</li><li>Pilot check valve</li><li>4/2 directional valve</li><li>E-spool directional valve</li></ul>	<ul style="list-style-type: none"><li>Logic valves</li><li>E-spool directional valve</li><li>Limit switches</li></ul>	<ul style="list-style-type: none"><li>Pressure control regenerative valve</li><li>G-spool directional valve</li></ul>	<ul style="list-style-type: none"><li>R-spool directional valve</li></ul>	<ul style="list-style-type: none"><li>E-spool directional valve</li><li>Check valves</li></ul>	<ul style="list-style-type: none"><li>Proportional directional valve</li><li>Check valve</li><li>Linear variable differential transformer (LVDT)</li></ul>	
A	Circuit characteristics	<div></div> $Q = \lim_{n \rightarrow \infty} \frac{Q_{\text{pump}} \left( \left( \frac{A_1}{A_2} \right)^n - 1 \right)}{\left( \frac{A_1}{A_2} - 1 \right)}, \quad v = \frac{Q_{\text{pump}} \times 10^5}{6(A_2 - A_1)}$ <p>Where <math>Q_{\text{pump}}</math> - pump delivery (l/min) <math>Q</math> - combined flow on full bore end (l/min) <math>A_1</math> - annulus area (mm<sup>2</sup>) <math>A_2</math> - piston area (mm<sup>2</sup>) <math>v</math> - extend velocity of the cylinder (mm/s) <math>n</math> - time step</p> <p>Remark: The output parameter <math>Q_{\text{pump}}</math> is an important message for finding the power supply sub-circuits</p>						
F	Type of control specified	Position control	Position control	Pre-set pressure control			Variable speed control with ram adjustment	
F	Max. allowable combined flow: Q (functional limit)	400 l/min	600 l/min	250 l/min	150 l/min	75 l/min	325 l/min	
F	Applications	<ul style="list-style-type: none"><li>High-low pressure presses</li><li>Injection moulding machines</li></ul>	<ul style="list-style-type: none"><li>High flow transfer</li><li>High flow press action</li><li>Injection moulding machines</li></ul>	<ul style="list-style-type: none"><li>High-low pressure presses</li></ul>	<ul style="list-style-type: none"><li>Fast transfer action</li></ul>	<ul style="list-style-type: none"><li>Fast transfer action</li></ul>	<ul style="list-style-type: none"><li>Speed control for transfer action</li><li>Injection moulding machines</li></ul>	
Inter	Interactions with other circuits/devices	No speed control sub-circuit						
		No connection with other circuits between this subsystem and actuator.	No load holding circuit.	No load holding circuit.			No pilot check load holding circuit is allowed.	

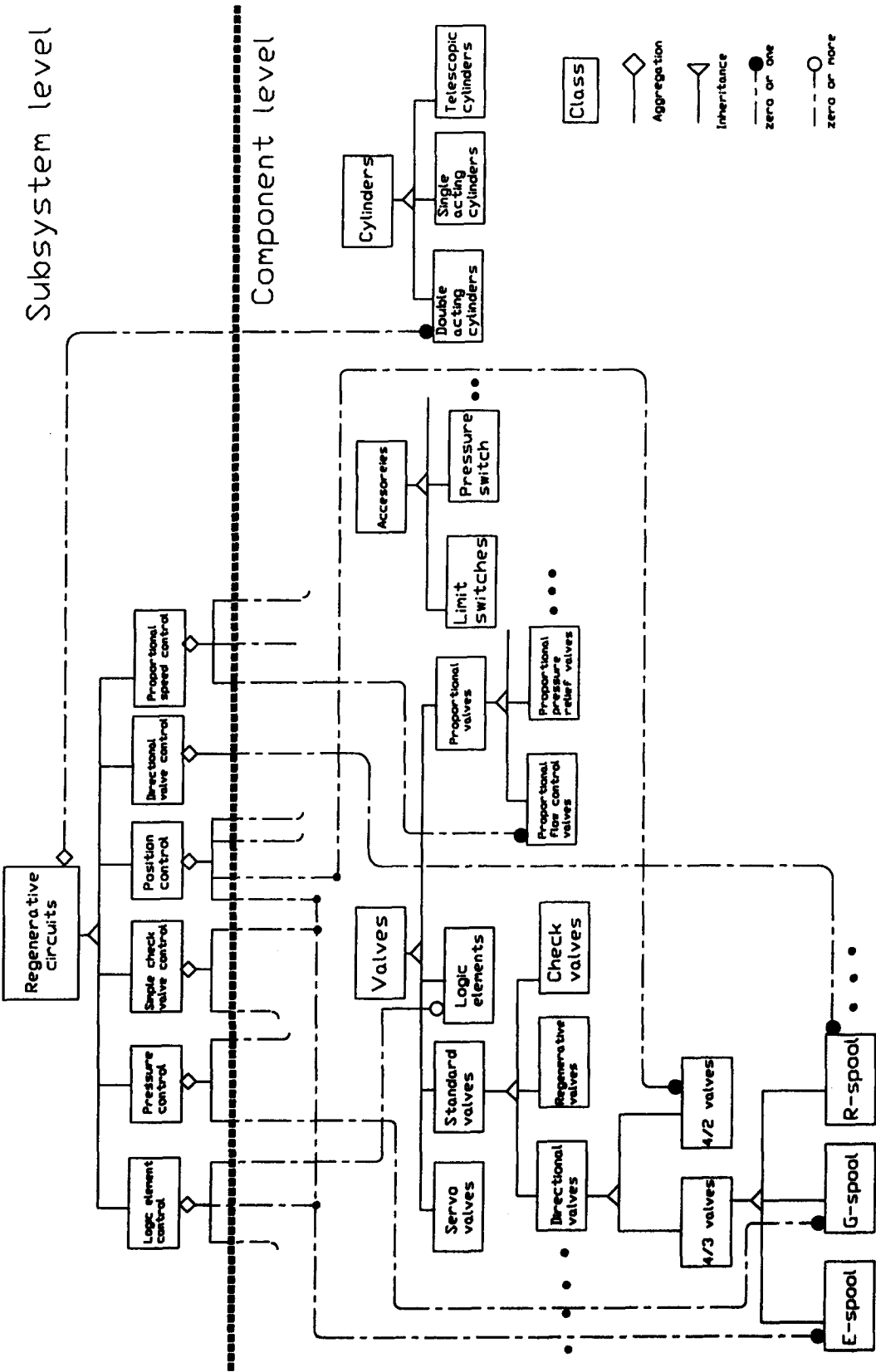


Fig. 3. Object-oriented data model of generic regenerative circuits.

capture (1) system parameters such as fluid type, flow rate, operating pressure, etc., (2) component-specific requirements, such as size, weight, cost, reliability, the names of symbols in drawings, etc., (3) component-specific functions, such as graphic display, algorithms for retrieval of suitable catalogue components from the object-oriented database, and (4) steady-state characteristics. At the stage of component selection, an appropriate object will be self-retrieved once it receives a message from its abstract class (generic class). Every abstract class encapsulates some methods that can determine suitable message paths to the subclasses, based on input parameters or messages. Different classes of components have various component sizing methods. For example: the theoretical bore size of a double-acting cylinder for push load with a return line pressure equal to the atmospheric pressure is determined by the following relationships:

$$D = \sqrt{\frac{4 \times F}{\pi \times P \times 0.95}}$$

where  $D$  is piston diameter (m),  $P$  is operating pressure ( $\text{N/m}^2$ ),  $F$  is maximum resultant load (N), and 0.95 is the guide value for the hydro-mechanical friction. However, for achieving equal extension and retraction speeds in a regenerative circuit, the ratio between the piston area and the rod area is equal to 2. If a regenerative circuit is considered for a transfer application, then the rod diameter for a single rod double acting cylinder is equal to  $F/P$ .

Then, the cylinder is checked against the operating pressure limit. Finally, the cylinder rod size is checked against the buckling limit, based on Euler's formula and the mounting method. In fact, the types and the sizing methods of cylinders are varied according to the specific classes of circuits, so that cylinder objects are usually considered to be one of the components of some sub-circuits. The sub-circuits also have pointers to link appropriate sizing methods in the cylinder classes.

In addition, some component objects contain performance data, serving as one of the criteria for the selection of catalogue components. The performance data usually refers to empirical operating curves supplied by the manufacturer's catalogues. Simple relationships can be described by appropriate equations, with coefficients found using the least-square method. However, for representing complicated isoefficiency lines, the Bézier–Bernstein approximation method<sup>15</sup> is selected for those three irregularly variable relationships. On the basis of this method, the complex characteristics can be stored, with the maximum and minimum values along the  $x$  and  $y$  axes, the  $x$  and  $y$  intervals, and the corresponding data along the  $z$  axis. Isoefficiency lines are always found in some components, such as variable-displacement pumps or hydraulic motors, and they are adopted to identify those operating conditions that are known to be necessary for higher efficiency and optimal utilisation.

Furthermore, every valve object also captures physical information for extending the application of hydraulic

circuitry design to system assembly design in another project. The physical attributes are the valve boundary, valve body size, port locations, port diameters, fixing hole locations and diameters.

### 3.4. Construction of the object-oriented model

Identification of the relationships and semantics of the identified objects is based on the working principles of the components and sub-systems. The advantages are: (i) Provision of a connection between functions and structures. Implementing a function by physical devices through an evaluation of the working principles allows the identification of promising classes of generic devices or sub-circuits without searching through a large number of catalogue components or sub-circuits. (ii) Enhanced readability of the system. (iii) Provision of a distinct framework for system maintenance and expansion.

In other words, sub-circuits or components with the same functions are grouped into a high-level module. These modules usually refer to an abstract class that contains many virtual functions and common properties. However, abstract classes would not create any object. Regenerative circuits and cylinders are examples of the abstract classes. The problem of the misuse of the override technique is likely to occur if object models are constructed without careful consideration. Consider the example in Fig. 4(a). Suppose a class of component or sub-circuit objects 1 which has attributes  $T1$  and functions  $F1$  ( $P1 < T1$ ,  $F1 >$  is called the property of a class/objects), and a class of component or sub-circuit objects 2 which possesses  $T2$  and  $F2$ . If two classes, say 1 and 2, have similar properties, the object model will generally be constructed in the following hierarchical form. An abstract class  $\alpha$  will be derived to share their common properties, so classes 1 and 2 become inherited classes of  $\alpha$  (i.e.  $\alpha = 1 \cap 2$ ), and are denoted as  $\delta_{1\alpha}$  and  $\delta_{2\alpha}$ .  $\delta_{1\alpha}$  and  $\delta_{2\alpha}$  contain some sets of additional properties of  $\delta P_{1\alpha} = \langle \delta T_{1\alpha}, \delta F_{1\alpha} \rangle$ ,  $\delta P_{2\alpha} = \langle \delta T_{2\alpha}, \delta F_{2\alpha} \rangle$  respectively. A new class of component or sub-circuit objects ADD is later created, where class ADD has similar properties to those of 1 and 2, plus its individual property  $\delta P_{(add)\alpha}$ . The flow function of ADD is different from classes 1 and 2, even though the function name is the same. Using the style in Eiffel,<sup>16</sup> the class ADD inherits class  $\alpha$ , and the differences are defined, leading to a redefinition of the flow function. Redefinition is achieved by means of function override in implementation [Fig. 4(b)]. The problem of this class structure is that, as an object in ADD is also a kind of object in  $\alpha$ , it might have two "flow characteristic" functions defined. In implementing the object-searching concept proposed in Section 2, it is extremely difficult for the computer to know which flow characteristic function is being referred to. The problem can be actually demonstrated in the C++ implementation (Fig. 5).

Implementation 2 is called when executing the problem. This is expected by a lot of programmers. However, in C++, implementation 1 is called instead. The problem of misuse of the override technique results in unreadable codes that discourage reusability. This kind of problem is always

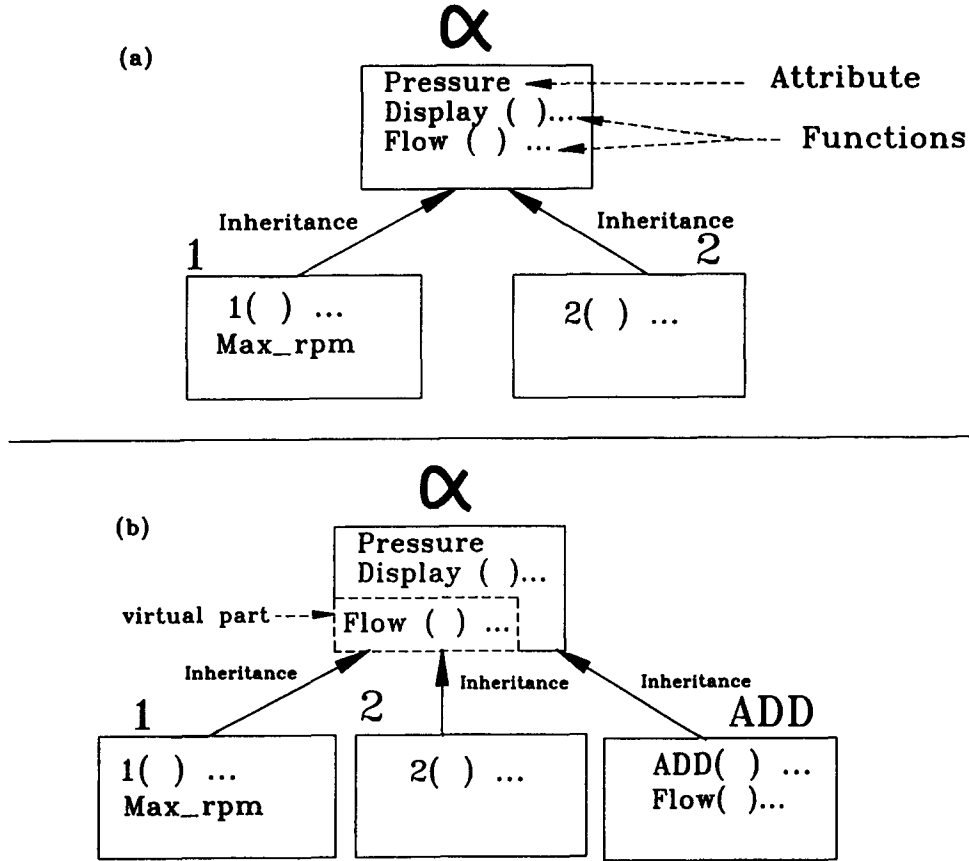


Fig. 4. Restructuring of the class structure where a new class ADD is later added.

neglected. A better class structure may be obtained by using the concept of class partitioning. Consider the above example; the class of  $\alpha$  may be partitioned into classes 1, 2 and ADD, where  $ADD = \alpha - 1 - 2$ . For achieving the concept, the original classes should be restructured using the following relations (mechanism):

$$\begin{aligned}
 T_{\alpha}(new) &= T_{\alpha} - \Delta T \\
 F_{\alpha}(new) &= F_{\alpha} - \Delta F \\
 \delta T_{1\alpha}(new) &= \delta T_{1\alpha} \cup \Delta T \\
 \delta F_{1\alpha}(new) &= \delta F_{1\alpha} \cup \Delta F \\
 \delta T_{2\alpha}(new) &= \delta T_{2\alpha} \cup \Delta T \\
 \delta F_{2\alpha}(new) &= \delta F_{2\alpha} \cup \Delta F \\
 &\vdots \\
 \delta T_{n\alpha}(new) &= \delta T_{n\alpha} \cup \Delta T \\
 \delta F_{n\alpha}(new) &= \delta F_{n\alpha} \cup \Delta F \\
 \delta T_{(add)\alpha}(new) &= \delta T_{(add)\alpha} \\
 \delta F_{(add)\alpha}(new) &= \delta F_{(add)\alpha}
 \end{aligned}$$

where  $\Delta T = (T_{\alpha} \cap \delta T_{(add)\alpha})$ ;  $\Delta F = (F_{\alpha} \cap \delta F_{(add)\alpha})$ ;  $T_{\alpha} = T_1 \cap T_2 \cap$

$\dots \cap T_n \cap T_{(add)}$ ;  $F_{\alpha} = F_1 \cap F_2 \cap \dots \cap F_n \cap F_{(add)}$  and the subscripts, 1, 2, 3, ...,  $n$ , represent class ID;  $\alpha$  represents an abstract class;  $(add)$  represents a new additional class.

The above restructuring mechanism for class expansion has been proved, fulfilling the following properties.

*Property 1: Invariance of functionality*

$$\begin{aligned}
 P_1 &= \{T_1, F_1\} \\
 &= \{T_{\alpha}(new) \cup \delta T_{1\alpha}(new), F_{\alpha}(new) \cup \delta F_{1\alpha}(new)\}; \\
 P_2 &= \{T_2, F_2\} \\
 &= \{T_{\alpha}(new) \cup \delta T_{2\alpha}(new), F_{\alpha}(new) \cup \delta F_{2\alpha}(new)\}; \\
 &\vdots \\
 P_n &= \{T_n, F_n\} \\
 &= \{T_{\alpha}(new) \cup \delta T_{n\alpha}(new), F_{\alpha}(new) \cup \delta F_{n\alpha}(new)\}; \\
 P_{(add)} &= \{T_{(add)}, F_{(add)}\} \\
 &= \{T_{\alpha}(new) \cup \delta T_{(add)\alpha}(new), F_{\alpha}(new) \cup \delta F_{(add)\alpha}(new)\}.
 \end{aligned}$$

*Property 2: Unique location of implementation*

$$\begin{aligned}
 P_{\alpha}(new) \cap \delta P_{1\alpha}(new) &= P_{\alpha}(new) \cap \delta P_{2\alpha}(new) \\
 &= \dots = P_{\alpha}(new) \cap \delta P_{n\alpha}(new) \\
 &= P_{(add)\alpha}(new) \cap \delta P_{(add)\alpha}(new) \\
 &= \emptyset.
 \end{aligned}$$



The purpose of Property 1 is to ensure that the functionality is not changed after restructuring. Property 2 is the unique location of functionality, which is very important because for the implementation of a class, each attribute or function is defined in one and only one class. Thus, the functionality of a class can be obtained by aggregating all functionalities in the derived class and the ancestors. By applying the above mechanism to the previous example, the error will be eliminated,

i.e.  $\Delta T = \{\text{pressure}\} \cap \emptyset = \emptyset$ ;  
 $\Delta F = \{\text{display}_0, \text{flow}_0\} \cap \{\text{ADD}_0, \text{flow}_0\} = \{\text{flow}_0\}$ ;  
 $T_\alpha(\text{new}) = \{\text{pressure}\} - \emptyset = \{\text{pressure}\}$ ;  
 $\delta T_{1\alpha}(\text{new}) = \{\text{max\_rpm}\} \cup \emptyset = \{\text{max\_rpm}\}$ ;  
 $\delta T_{2\alpha}(\text{new}) = \emptyset \cup \emptyset = \emptyset$ ;  
 $\delta T_{(\text{add})\alpha}(\text{new}) = \emptyset$ ;  
 $F_\alpha(\text{new}) = \{\text{display}_0, \text{flow}_0\} - \{\text{flow}_0\} = \{\text{display}_0\}$ ;  
 $\delta F_{1\alpha}(\text{new}) = \{1_0\} \cup \{\text{flow}_0\} = \{1_0, \text{flow}_0\}$ ;

$$\delta F_{2\alpha}(\text{new}) = \{2_0\} \cup \{\text{flow}_0\} = \{2_0, \text{flow}_0\};$$

$$\delta F_{(\text{add})\alpha}(\text{new}) = \{\text{ADD}_0, \text{flow}_0\}.$$

The implementation method for the previous example is just to change the codes in class  $\alpha$  as shown below.

```
class  $\alpha$ 
{ :
  virtual float flow( ) {return (pressure*1.3972);} //
    Implementation 1 of flow characteristic function
  :};
```

Reviewing the regenerative circuit model presented in Fig. 3, valves are derived into standard, logic, servo and proportional valves. By applying the above class-partitioning method, the class of valves may be easily identified without any confusion, and it may be regarded as an abstract class which is equal to standard valves+logic valves+servo valves+proportional valves instead of using the concept of

```
class  $\alpha$ 
{ public:
  int pressure;
  void display( ) {...};
  float flow( ) {return (pressure * 1.3972);} // Implementation 1 of flow characteristic function
  :
};
class 1 : public  $\alpha$ 
{ int max_rpm;
  public:
  1( ) { pressure = 100; .....} // constructor for the object 1 with default pressure limit 100 bar
  :
};
class 2: public  $\alpha$ 
{ public:
  2( ) (pressure = 140; .....} // constructor for the object 2 with default pressure limit 140 bar
  :
};
class ADD :public  $\alpha$ 
{public:
  ADD( ){pressure = 210; .....} // constructor for the object ADD with default pressure limit 210 bar
  float flow( ) {return( pow( pressure, 2 ) * 2.583);} //Implementation 2 of flow characteristic function
  :
};
main( )
{
   $\alpha$  *p[2]; // searching pointer of this module and it belongs to abstract class
  p[0] = new 1( ); p[1] = new 2( ); p[2] = new ADD( ); //construct the objects
  :
  for(int i=0; i<3; i++) // search from the first layer
  { if (p[i]->pressure > 200) //Find a component which pressure limit is bigger than 200 bar
    p[i]->flow( ); // Implementation 1 involved, this is an unexpected implementation
  }
}
```

Fig. 5. Example of calling an unexpected implementation of a function using C++.

subclasses (i.e., servo valves are typically thought of as one kind of valves).

For some subsystems, their functional relationships are not clear, so it is difficult to organise the hydraulic subsystems in a class hierarchy. However, the problem can be solved if the hydraulic subsystems are viewed as modifications of one or more simple types of subsystem. For example: a dual-pump circuit may be regarded as a modification of a single-pump supply circuit. It may be more direct to form the object model by means of inheritance. The general approach is to take the subsystem involving the elementary/most-common/minimum number of components to be the top class, and then the modification of the simple subsystem is the derived class. However, the problem of misuse of the override technique also occurs in this structure. Thus, not every function may be uniquely defined. Another problem is that every hydraulic subsystem is an aggregation of primitive objects (hydraulic components). In cases where some composite objects of the top class are different from their subclasses, their subclasses are going to inherit some unwanted composite objects, and as a result will create erroneous objects. Figure 6 describes two

sub-circuit classes, A and B, where class B is considered as a modification of class A. In addition, A and B are composite objects of "component\_X" and "component\_Y" objects and "component\_X" and "component\_Z" objects respectively; the two classes also have the same *draw* functions but different versions of *display* functions. Figure 6(a) shows that class B inherits an unwanted component object X in such a class structure. These problems are actually found in C++ implementations as well. A better class structure can also be obtained by using the concept of class partitioning described above. The improved class structure [see the same example in Fig. 6(b)] can be obtained using the following relations (mechanism):

$$T_c = T_a - (T_a \cap \delta T_{ba})$$

$$F_c = F_a - (F_a \cap \delta F_{ba})$$

$$\delta T_{ac} = T_a \cap \delta T_{ba}$$

$$\delta F_{ac} = F_a \cap \delta F_{ba}$$

$$\delta T_{bc} = \delta T_{ba}$$

$$\delta F_{bc} = \delta F_{ba} \quad c \text{—a new abstract class}$$

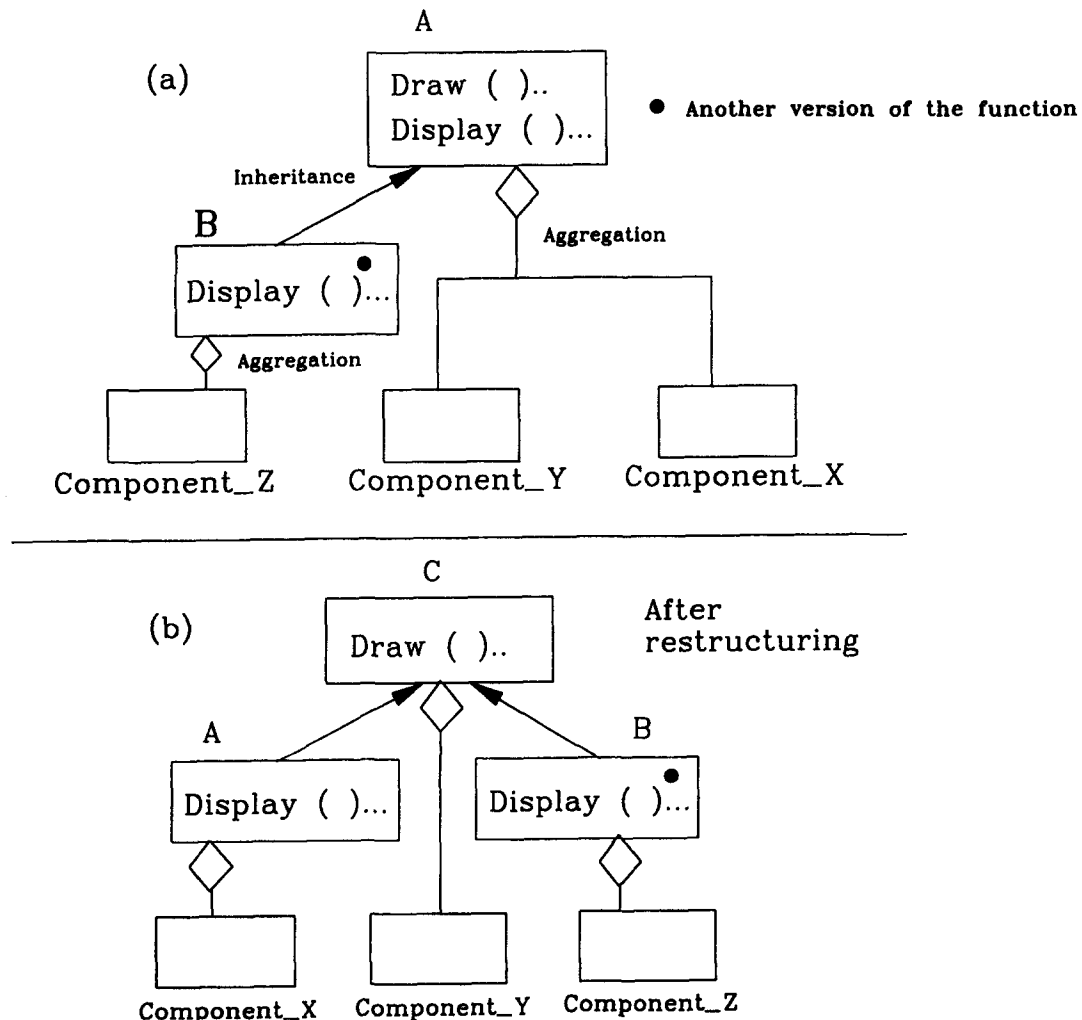


Fig. 6. Restructuring of the class structure where class B is the modification of class A.

The above class relations also fulfil the properties of:

- (1) Invariance of functionality  
 $i.e. \Rightarrow P_a = \{T_c \cup \delta T_{ac}, F_c \cup \delta F_{ac}\}$   
 $P_b = \{T_c \cup \delta T_{bc}, F_c \cup \delta F_{bc}\}$
- (2) Unique location of implementation  
 $i.e. \Rightarrow T_c \cap \delta T_{ac} = \emptyset, F_c \cap \delta F_{ac} = \emptyset$   
 $T_c \cap \delta T_{bc} = \emptyset, F_c \cap \delta F_{bc} = \emptyset.$

By applying the above methodology, there may be more classes after restructuring. However, the relationships among these classes are much clearer. Furthermore, the functionalities of a class can be obtained very easily by aggregating the functionalities of the ancestor classes. As each functionality only appears in one and only one class, the location of implementing the functionality and its details can be obtained easily. It is believed that the above pieces of work are able to serve as references for object-oriented design, and provide a potential implementation mechanism for automatic class reconstruction. Nevertheless, the readability and semantic meaning of the system should be considered, in order to avoid the generation of some difficult-to-understand classes.

#### 4. A BRIEF OVERVIEW OF THE CAD SYSTEM ARCHITECTURE

The CAD-system architecture is shown in Fig. 7. It includes an OODB, an object server for object manipulation and management, and a circuit design program. The program contains an inference mechanism integrating a knowledge-base object, which stores application-specific rules for circuit design. The purpose of these rules is to deduce hydraulic system requirements from the design specifications of different applications. For instance: if a deep-drawing press is designed, then the load type of the actuators is set to "vertical push load", and a three-phase motor is set to be the prime mover. The separation of the general hydraulic system design knowledge and the application-dependent knowledge results in the flexibility of the CAD system. Changes of hydraulic applications only require changes of the application-specific knowledge-base module. The persistent object storage is for catalogue components which link to the component level of the data model.

On the basis of the philosophy proposed in Section 2, the circuit design system can automatically synthesise hydraulic

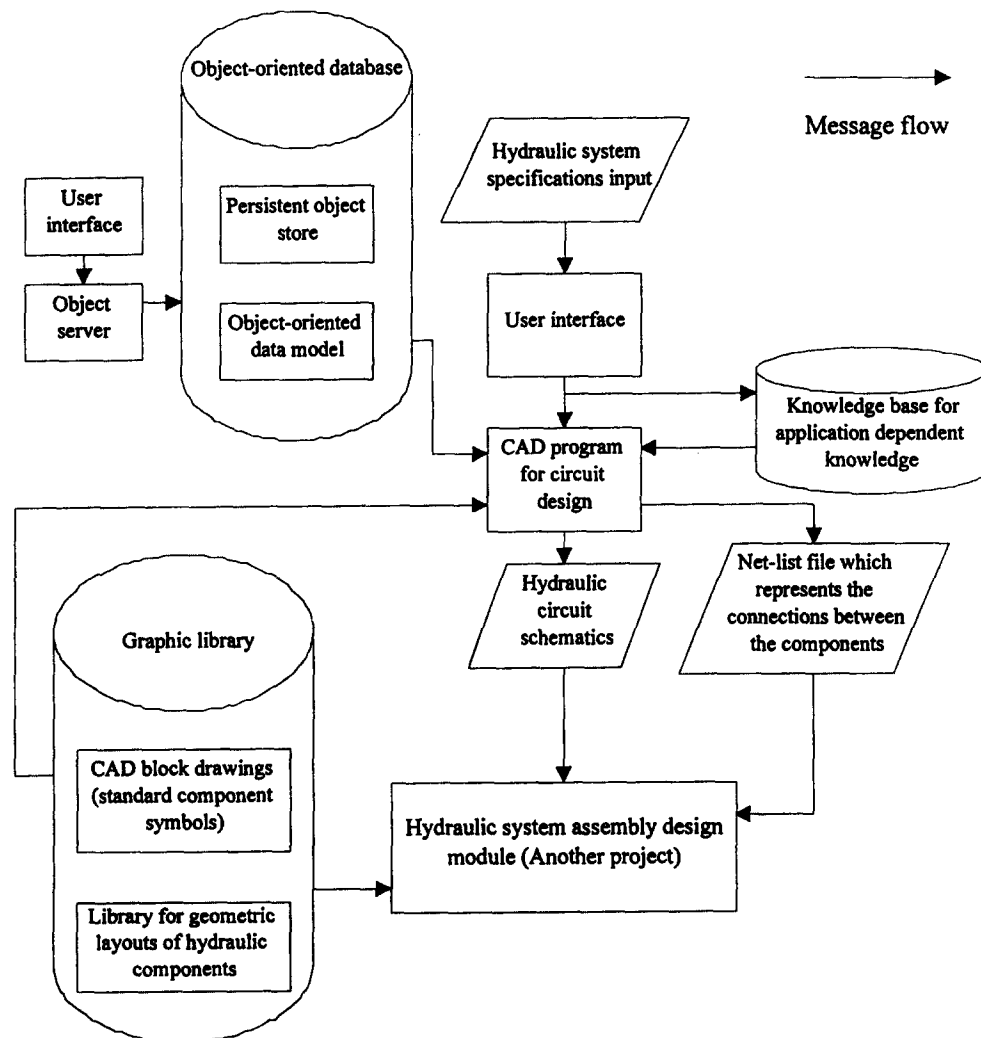


Fig. 7. The structure of the proposed CAD system.

circuits. The circuit design process starts from the design of the output drive, followed by the establishment of the power flow and signal flow, and selection of the control units and power units. A detailed description of the hydraulic circuit design procedures is shown in Fig. 8. In the design of a hydraulic system, the maximum operating pressure of a

system would be defined according to the type of application. The pre-set value is retrieved from the application-specific knowledge-base.

Users are asked to define the numbers of groups of output required and the essential specifications of the actuators, both the hydraulic cylinders and/or the hydraulic motors,

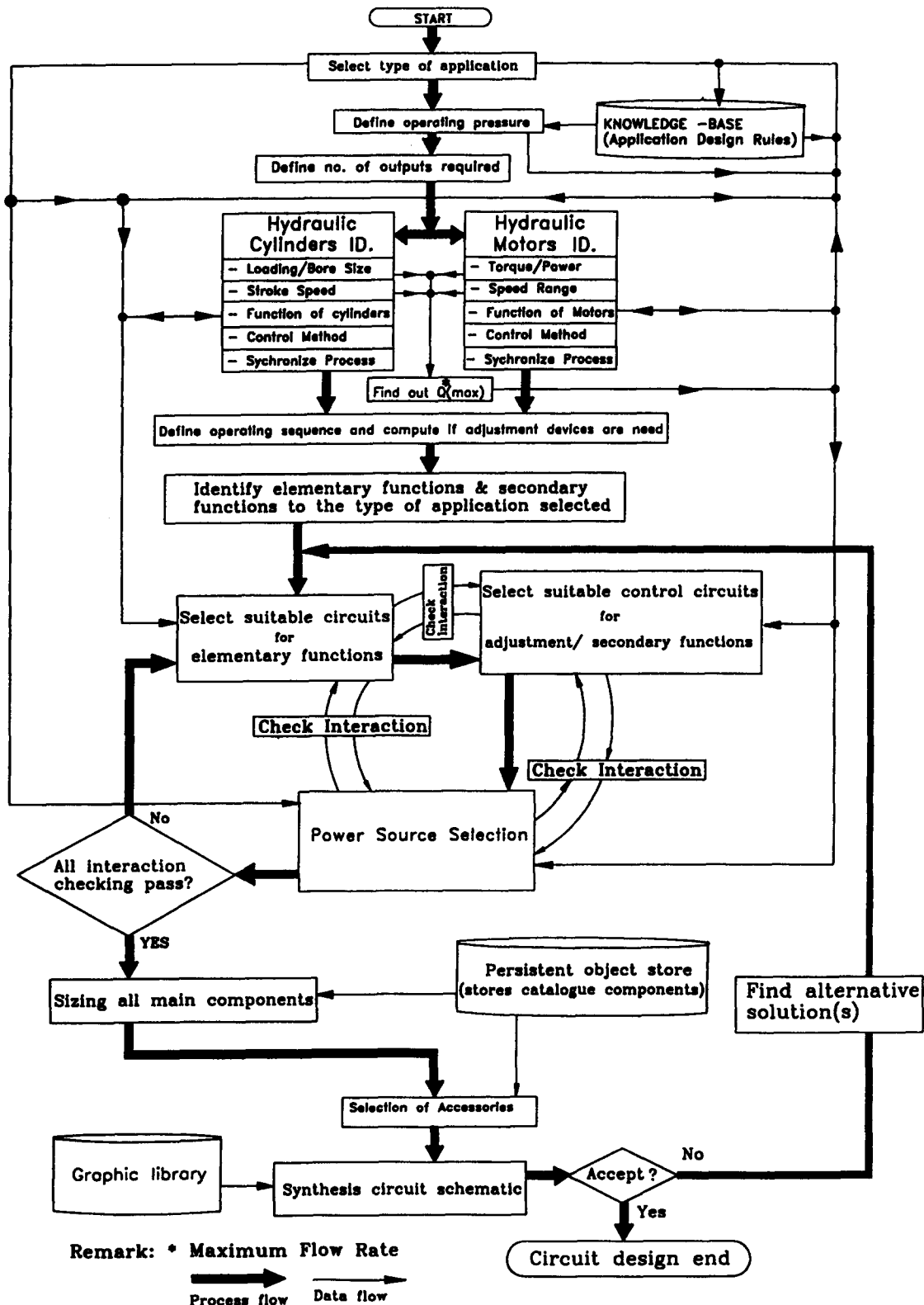


Fig. 8. The process chart of the CAD system for designing general hydraulic circuits.

through a user interface. The load of each cylinder with the stroke speed, and the torque of each motor with the rotary speed, are the key parameters for calculating the maximum flow rate of the system. Furthermore, the designers are sometimes required to input the motion sequence of the actuators, and to indicate the groups of actuators for simultaneous paths if necessary. The main control of each actuator is retrieved from the object-oriented database, based on matching the specifications of the functionalities ( $F$ ) of the presynthesized subsystems with the system requirements, next to the selection of adjustment control, if necessary. Adjustment control (such as pressure and flow control) would supplement the backbone of the circuit. The selection of a pump source is driven by the features of the selected primary functions of the sub-circuits. Different pumps have their own pressure and flow features; it is reasonable to combine the pumps together to form a main pump source unit for a different supply. It may be observed from Fig. 8 that there is a cyclic interaction checking process between each pair of sub-function design modules, for ensuring that every sub-circuit/device selected can be linked without contradiction and for eliminating duplicate functional control. Users are sometimes required to make decisions for alternative choices in the design process, such as the use of a pressure meter, the types of the actuation interfaces of each valves; the availability of these optional choices largely depends on the sub-circuits retrieved.

Afterwards, all the control valves would be sized and selected according to the flow rate and the operating pressure of the system. The actual model numbers and dimensions of the valves are defined as the persistent objects which are stored in the database. The last step is to determine the accessories such as oil tank, cooling system and filtration system, etc. Then, a complete schematic circuit diagram would be generated with the aid of a graphic library containing sub-circuit and component block drawings.

The circuit design program also contains a schematic capture module for capturing the connection information from the schematic and specified hydraulic components. The information is in netlist format for input to another project, hydraulic system assembly design.

## 5. APPLICATION EXAMPLE

A prototype CAD system for the class of hydraulic presses, written in C++ and running on an AutoCAD platform, has been built to implement the concepts described in this paper. Hydraulic presses can be divided into many types, such as forging, blending, drawing, etc. Every press has its individual standard hydraulic configuration, plus some common shareable existing sub-circuits. This section illustrates the prototype CAD system used for the design of a practical industrial hydraulic system. To reduce the complexity of the CAD system, sub-circuits and components are implemented by means of simple codes to represent the object identities, class names and the attribute values, in order to facilitate the symbolic matching process. The system, as implemented, is able to recommend both an appropriate hydraulic circuit and the size of the components, once the CAD system accepts an allowable design specification. The input specifications for an example of a 50-ton deep-drawing hydraulic press are shown in Table 2.

### 5.1. Discussion of the results

Figure 9 shows the circuit configuration and main component size recommended by the CAD system. In this example, the main actuator is first set to vertical push load, and holding load, fast forward control and low-high pressure control are default functions. Flow control and pressure reduction are adjustment functions for achieving the desired performance of the cylinders, since a deep-drawing press has at least two cylinders at different working pressures, supplied by one pump source.

Table 2. The input specifications of a 50-ton hydraulic deep-drawing press

Cylinder name	A (ram cylinder)	B (die cushion cylinder)
Maximum loading (ton)	50	26
Cylinder constraint (bore diameter/mm)	nil	nil
Cylinder constraint (rod diameter/mm)	nil	nil
Stroke length (mm)	500	300
Max. stroke speed (mm/sec)	63.5	63.5
Max. stroke speed for loading (mm/sec)	10	10
Mounting method	front	back
Actuator function	pressing	support
Variable force control	nil	nil
Type of control in press action	position	position
Variable speed control	nil	nil
Working with simultaneous paths	nil	nil

Mould ejector: nil

Operation hour per day: 24 hours

\*Default machine sequence for deep drawing presses without ejector: A+,

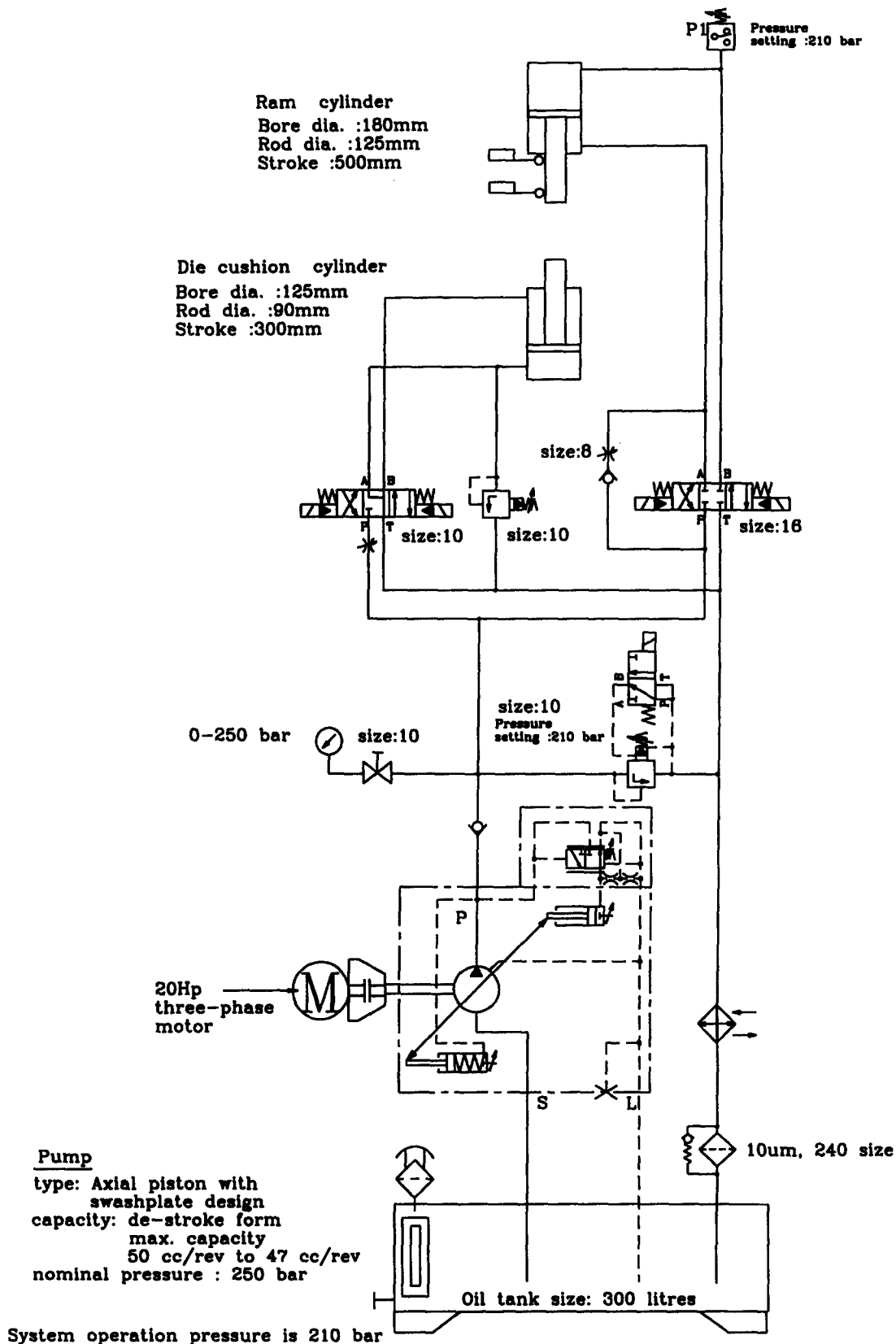
P1: T1B-, A-, B+

Remarks:	Tn	time delay (where $n=0, 1, 2, \dots$ )
	P1	pressure switch
	+	actuator traverse forward
	-	actuator traverse backward
	( ): ( )	trigger signal: motion



modules. Regarding the main component sizes in this application, the cylinder types and sizes are first inferred, subsequent to power supply unit. The internal and external

flow-rate distributions in each sub-circuit selected are calculated, based on their defined circuit characteristics, and these parameters serve as messages to pass with each other.



**Fig. 10. Hydraulic circuit for the deep-drawing press designed by an experienced engineer.**

These messages, plus the operating pressure limit, are very important in the selection of the internal component sizes and the other sub-systems. Referring to Fig. 9 again, the external output flow rate of the selected position control regenerative circuit is used for selection of the pump circuits. Its internal flow rate is used for determining the sizes of the valves RGV1, RGV2 and RGV3, and each selected valve has been checked against the permissible flow limit at the calculated system parameters, using individual encapsulated empirical performance data.

The results generated from the prototype CAD system have been built, and were verified to perform in accordance with the stipulated specifications by a hydraulic engineering company. The results have also been compared with the design in Fig. 10, which was designed by an experienced engineer and has actually been adopted in industry. The number of components used in Fig. 10 is more than that used in Fig. 9. The design in Fig. 9 adopted regenerative circuits to provide high-speed motion associated with energy-saving considerations, whereas the design in Fig. 10 only selected a large-sized pump with low-high pressure control, resulting in large sizes of the tank and prime mover. Thus, the design in Fig. 9 is better than that in Fig. 10, in terms of long-term energy-saving and component cost.

In addition to improving the qualities of both design and draft, another contribution of this CAD system is the reduction of design lead time. Non-experts normally spend one to two months on completing this type of design, whereas the CAD system only requires a few minutes. The major difficulty in manual design is to find appropriate and effective sub-circuits and components from a lot of catalogues and handbooks, particularly in the selection of sub-systems. This is because the definitions of sub-systems are not standardised, and finding and understanding the properties of each effective candidate sub-circuit is a time-consuming task, even for experts. In addition to that, this CAD system is not confined to a specific hydraulic application. Using object-oriented programming principles, the applications can be changed and additions and modifications to the components and existing sub-circuits can be easily incorporated into the program with minimal modifications. In fact, the objects used in this prototype system have reused some objects developed<sup>17</sup> for pneumatic sequential circuit design carried out by the authors. Moreover, the object-oriented approach proposed in this project can provide hydraulic engineers with more systematic thinking than a rule-based format, in formalizing the design knowledge.

## 6. CONCLUSIONS

This work has drawn on a mixture of AI, object-oriented technology and CAD techniques for the building of systems with the ability to solve configuration design problems. The proposed object-oriented approach is a new methodology for circuit design. It has been proved to work successfully,

and is quite efficient in the complex hydraulic application domain, if the concept of dynamic modules is adopted for identifying hydraulic subsystem objects. The component details considered and recommended in this CAD system are more extensive than in other expert systems for hydraulic applications, found in the literature. Moreover, the class-reconstruction principles enable the class structure of hydraulic sub-circuits and components to be defined without error. The CAD system will be further extended to deal with fluid power-system assembly problems.

**Acknowledgements**—The authors would like to thank staff of Mannesmann Rexroth (China) Ltd. for their help during the verification of the system. This research is supported by The Hong Kong Polytechnic University Research Grant No. 341/165.

## REFERENCES

- Burton, R. T and Sargent, C. M., The use of expert systems in the design of single and multi-load circuits. *Proceedings of the 2nd International Conference on Fluid Power Transmission and Control*, 1989, pp. 605–610.
- Li Congxin, Huang Shuhuai and Wang Yungan, An expert system for designing hydraulic schemes. *Proceedings of the 2nd International Conference on Fluid Power Transmission and Control*, 1989, pp. 611–616.
- Changchun Zhao, Ying Chen, Li Xu, Yongxiang Lu, An ICAD system for plastic injection moulding machine. *Proceedings of 3rd the International Conference on Fluid Power Transmission and Control*, 1993, pp. 369–376.
- Kong, K. K., Chuen, C. W., Wong, T. T. and Leung, T. P., An integrated approach in implementing hydraulic expert systems for both design and diagnosis. *Proceedings of the 3rd International Conference on Fluid Power Transmission and Control*, 1993, pp. 381–387.
- Kota, S. and Lee, C., A look at an expert system for hydraulic design. *Hydraulic and Pneumatics*, 1990, **43**(5), 48–51, 71.
- Cox, B. J., *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, MA, 1986.
- Zhang Xiaoping, Object-oriented knowledgebase application in hydraulic design. Master's thesis, University of Florida, 1991.
- Akagi, S. and Fujita, K., Building an expert system for engineering design based on the object-oriented knowledge representation concept. *ASME of Mechanical Design*, 1990, **112**, 215–222.
- Philip, C.-Y. Sheu, VLSI design with object-oriented knowledge bases. *Computer-Aided Design*, 1988, **20**(5), 272–279.
- Lau, S. K., Leung, T. P., Wong, C. M., An object-oriented knowledge based approach for variational mechanism design. *International Manufacturing Conference with China, Proceedings of IMCC'93*, 1993, **2**, 371–377.
- Coad, P. and Yourdon, E., *Object-oriented design*. Yourdon Press, 1990.
- Booch, G., *Object-oriented design with applications*. Benjamin Cummings, Redwood City, CA, 1990.
- Rumbaugh, J., *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- The Object Agency Inc., A comparison of object-oriented development methodologies, 1992–1993 report.
- Plastock, R. A. and Kalley, G., *Theory and Problems of Computer Graphics*, Schaum's Outline Series in Computers. McGraw-Hill, New York, 1986, pp. 180–181.
- Meyer, B., *Object-Oriented Software Construction*. Prentice Hall, Englewood Cliffs, NJ, 1988.
- Wong, P. K., Leung, T. P., Chuen, C. W. and Chan, W. H., Object-oriented CAD for electro-pneumatic sequential circuit design in low cost automation. *Proceedings of SEIKEN/IEEE Symposium on Emerging Technologies and Factory Automation*, Japan, 1994, pp. 278–284.