AUTOMATION IN DESIGN: THE CONCEPTUAL SYNTHESIS OF CHEMICAL PROCESSING SCHEMES

Chonghun Han and George Stephanopoulos

Laboratory for Intelligent Systems in Process Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139

James M. Douglas

Department of Chemical Engineering
University of Massachusetts, Amherst, MA 01003

If you really know how to carry out an engineering task, then you can instruct the computer to do it automatically. This self-evident truism can be used as litmus test of whether a human "really" knows how to, say, design an engineering artifact. Experience has shown that in very few instances, engineers have been able to automate the process of design, thus demonstrating the presence of serious flaws in (1) their understanding of how to do design or/and (2) their ability to clearly articulate the design methodology, both of which can be traced to the inherent difficulty of making the "best" design decisions. The pivotal element in automating the design process is *modeling the design process itself*, which includes the following modeling tasks: (1) modeling the *structure of design tasks* that can take you from the initial design specifications to the final engineering artifact; (2) representing the *design decisions* involved in each task, along with the assumptions, simplifications, and methodologies, needed to frame and make the design decisions; and (3) modeling the *state of the evolving design*, along with the underlying rationale.

In this chapter, we will show how one can use ideas and techniques from artificial intelligence, such as symbolic modeling, knowledge-based systems and logic, to construct a computer-implemented model of the design process. By using the Douglas hierarchical approach as the conceptual model of the design process itself, this chapter will show how to generate models of the structure of design tasks, design decisions, and the state of design, thus leading to automation of large segments of the synthesis of chemical processing schemes. The result is a *human-aided, machine-based* design paradigm, with the computer "knowing" how the design is done, what the scope of design is, and how to provide explanations and the rationale for the design decisions and the resulting final design. Such paradigm is in sharp contrast with the traditional *computer-aided, human-based* prototype, where the computer carries out numerical calculations and data fetching from files and databases, but it has no notion on how the design is done, i.e., knowledge resting exclusively in the province of the individual human designer. In addition, we will argue that the human-aided, machine-based design is the paradigm that will characterize future design systems, where rapid conceptualization and prototyping of engineering artifacts will be the source of competitive edge.

I. Introduction

The engineering design of products and/or processing systems is a dialectic process (Stefik *et al.*, 1982) between goals (i.e., what is desired) and possibilities (i.e., what is actually realizable), aimed at the satisfaction of functional and performance specifications. No general theory exists for the systematic and rigorous development of a procedure that leads to the design of the desired engineering artifacts. This is due to two inescapable facts: (1) any design is a knowledge-intensive task; the more and better-quality knowledge, the more efficient the design and better the quality of the designed artifact and (2) the knowledge required for a particular class of design problems is specific to the class of problems. Thus, various attempts to formalize the overall design procedure as a large combinatorial optimization problem, have not provided a *generic theoretical framework for design*, but have simply underlined the fact that every design problem is a complex decision process with real-valued and integer decisions. On the other hand, combinatorial optimization techniques based on branch-and bound strategies (Edgar and Himmelblau, 1988), when applied to narrowly defined specific domains, have been shown to be fairly effective in selecting good design solutions through the implicit enumeration of many alternatives (Grossmann, 1985, 1989; Kocis and Grossmann, 1989). Nevertheless, even in such cases, the successes are an indication of the efficiency of the implicit enumeration techniques rather than a manifestation of a theory for design.

In the absence of a general theory on how design is done, research in the filed of artificial intelligence has been addressing the following distinct but complementary areas of inquiry:

1. *Axiomatic theory of design*, with the objective to establish a theoretically firm ground for the definition of design and thus bring it into the realm of "science," rather than "art," where it presently stands. Efforts in this direction have led to a number of approaches for the representation of cognitive models of the design process, but no significant breakthroughs have been achieved.

2. *Engineering science of knowledge-based design* (Tong, 1987; Tong and Sriram, 1992), aiming at the development of a rational framework for organizing, evaluating and formulating knowledge-based models on how the design is done. Efforts in this direction have been more successful. They have led to a number of new representation schemes (e.g., frames, scripts, streams, actors; see Rich and Knight, 1991; Quantrille and Liu, 1991), which have broken previous limitations in articulating, representing, and utilizing all forms of available knowledge; a particularly important requirement for a successful design approach. In addition, theoretical work has led to the development of new mathematical frameworks, which can manipulate these forms of knowledge representation (e.g., algebra of approximations, logic inferencing through semantic networks, analysis of logic clauses). Consequently, today we can effectively use all forms of

available knowledge within a computer-based environment; this is a significant addition to the numerical algorithms of the traditional computer-aided design tools.

Computer-aided design environments should support the designer in reaching better solutions than in the past, but in shorter periods of time. *We cannot afford to have the designs of tomorrow take as long as the designs 20 years ago.* Here is where the new artificial intelligence (AI)-based programming styles (e.g. object-oriented programming), the development of design-oriented languages, and object-oriented databases have already made significant contributions to the design process, by allowing the development of highly complex, design-oriented software systems in remarkably short periods of time. The existing applications, especially in other areas of engineering, are very impressive and do not allow room for disputing this fact. In particular, the complexity of the so-called *cooperative design environments* (or equivalently, *concurrent engineering*) is of such scale that traditional procedural programming approaches would have rendered the development of such software systems impossible. Thus, *AI is enabling the development of new design environments.* In chemical process design, the corresponding effort in academic institutions is at early stages, although the articulation of the design needs by the industry has been expressed in more mature terms. Lack of education in the developments of modern computer science has been the primary reason for this delay.

A. Conceptual Design of Chemical Processing Schemes

Conceptual process design involves the series of tasks leading to the development of a process flowsheet from the given reaction information and product specifications. This process usually comes at the early stage of a design project. Since all other design activities depend on the results of the conceptual process design, it has been considered the most important stage during the development and deployment of a new process. According to Douglas (1988), while the cost of the conceptual design usually takes 10–20% of the total cost to develop a new commercial process, the decisions committed at this stage fix about 80% of the total project cost.

Conceptual process design is an underdefined problem. Only a very small fraction of the information needed to define a design problem is available from the problem statement. The design decisions of the process designer provide this missing information. For example, the designer makes design decisions about what kind of process units to use, how to interconnect those units, and so on. From this perspective, the conceptual design is a combinatorial problem. It has been reported (Douglas, 1988) that there are 10^4–10^9 ways that we might consider to accomplish the conceptual design. On the other hand, experience indicates that at the conceptual design level, less than 1% of ideas ever prove to be successful, so the emphasis should be placed on the quick screening of numerous process alternatives.

1. Previous Approaches and Their Limitations

The pivotal role of the conceptual process design has motivated significant levels of research on the automation of the conceptual process design. AIDES (Adaptive Initial DEsign Synthesis) (Siirola *et al.*, 1971; Powers, 1972; Rudd *et al.*, 1973) is the pioneering prototype in the automatic synthesis of conceptual process flowsheets. AIDES makes use of a heuristically modified version of the means–ends analysis technique (Quantrille and Liu, 1991, pp. 268–270); given information on raw materials and the desired products, it attempts to bridge the difference between raw material and products through a sequence of operations that are eventually transformed into unit operations. BALTAZAR (Mahalec and Motard, 1977) is another early example on the use of AI techniques in process synthesis. Using techniques from the area of automatic theorem proving, BALTAZAR proposes logic-based approaches to the synthesis of flowsheets.

Douglas (1988) has proposed a hierarchical decision procedure. This procedure decomposes the design problem into a series of design or decision levels, sequenced and solved in a hierarchical order. It first sketches a design that is complete but vague, and then refines the vague parts into more detailed designs until finally the design has been refined to a complete sequence of detailed unit operations. Heuristics are used at all levels to fix the structure of the flowsheet. Economic potential is computed at each level to reduce the number of alternatives that have to be considered. It should be noted that hierarchial decision procedure is a systematic design procedure, not a design automation system.

Synthesis of process flowsheets through a simultaneous structural and parametric optimization, has received significant levels of attention (Grossmann, 1985, 1989; Kocis and Grossmann, 1989). This approach requires a superstructure as an initial structure from which all the designs of interest can be derived. Branch-and-bound algorithms have been used to solve the mixed-integer nonlinear programming (MINLP) formulations, and find the optimal structure and parameters of the unknown flowsheet. The possible design alternatives need to be articulated and modeled, if one is to develop an MINLP formulation that, using an implicit enumeration algorithm, will solve the MINLP problem and find the optimal solution. Clearly, this approach does not give much attention on the design process that we want to automate. However, it is a very promising technique for selecting the best alternative, once the superstructure has been identified.

Knowledge-based expert-system development has been very active during the last decade. After a design methodology has been identified for a given class of design problems, the expert system can be constructed in such a way that it maps uniquely the engineering design methodology into a computer program. The human usually interacts with the program by monitoring the design process, providing decisions and guidance at critical junctures, or the values which the program requests. Example of such programs are *PIP* (Kirkwood, 1987; Kirkwood *et al.*, 1988), *BioSepDesigner* (Siletti, 1988; Siletti and Stephanopoulos, 1992), *ProDesigner* (Kritikos, 1991). The construction of PIP (a program for the synthesis of process flowsheets) led to a rather rigid program with a strict procedural model to emulate the design methodology. Currently, a new object-oriented version is under development with significantly improved flexibility, extensibility and maintenance. *BioSepDesigner*, on the other hand, was developed to be a flexible design environment for the synthesis of separation sequences for the recovery and purification of proteins. Object-oriented in character, it is easily extendible with new knowledge and provides integrated treatment of data, models, and design decisions. Nevertheless, all three of the above systems and several others do not contain an explicit model of the design process and thus they cannot easily accommodate the incorporation of new knowldege as it becomes available.

B. Issues in the Automation of Conceptual Process Design

Extensive efforts over the last 30 years have led to rich computer-aided design environments that support the various tasks during the synthesis, development, and engineering of processing schemes. Figure 1 shows the standard outlay of such a computer-aided design (CAD) environment. A series of tools (e.g., equation solvers, optimization routines, specific design methodologies for localized problems, physical property estimation techniques) are being invoked by the human designer to execute various design tasks, such as design of heat exchanger networks, synthesis of separation systems, design of individual processing units, sizing, and costing. Convenient graphic interfaces make the interaction between designer and computer very smooth and transparent. Sophisticated database management

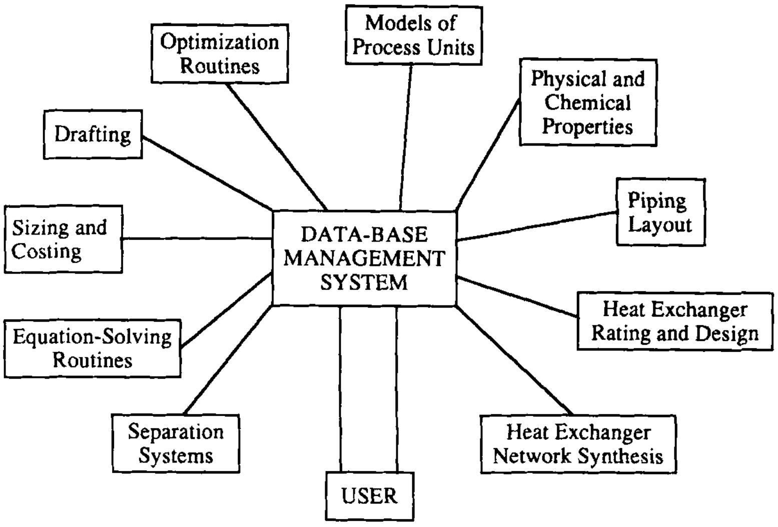

Optimization Routines
Models of Process Units
Physical and Chemical Properties
Drafting
Piping Layout
Sizing and Costing
DATA-BASE MANAGEMENT SYSTEM
Heat Exchanger Rating and Design
Equation-Solving Routines
Separation Systems
Heat Exchanger Network Synthesis
USER

FIG. 1. Typical computer-aided design environment.

systems provide easy means for depositing and retrieving of data, intermediate results, previous designs, etc.

Despite the continuous enrichment of CAD environments, the character of the overall process design procedure remains the same (Mostow, 1985): "the human does the design and the computer provides the support tools, without understanding the design process, its rationale, or the design decisions." The drawbacks of the CAD paradigm are several and, at times, critical. They all stem from the fact that the design procedure is implicit in the designer's mind, for example

(a) Where does a design start from?
(b) What is to be done next?
(c) How are the assumptions, conjectures, and simplifications needed for the design to proceed, to be formulated?
(d) How are major design decisions made?

In other words, the CAD paradigm "knows" nothing about (1) the design agenda and (2) the "context" of the design activities.

Furthermore, the computer structure of tasks during the synthesis of a process flowsheet can be very large, detailed and complex for any human designer to document and mentally carry with him/her. To the extent that we can untangle and make explicit the design procedure, thus emulating the designer's own methodology, the process can be mechanized. But in this case, we are moving towards a *human-aided, machine-*

based design paradigm, where the computer, through human guidance, can carry out significant portions of a design by "knowing" the design procedure itself, its rationale and the reasoning behind a number of design decisions. This is the paradigm whose development and computer implementation has been significantly advanced by research in artificial intelligence, and which we will discuss in subsequent sections. The benefits from the availability of such mechanized models for design are many and diverse:

1. Rapid prototyping and evaluation of processing schemes.
2. Improvements in cost and reliability.
3. Explicit documentation of the design process itself: why certain goals were set during the design and how they were achieved, how design decisions were made, what assumptions and simplifications were involved, what models were used at the various stages of design, what alternative designs were examined, and why certain ones were selected over others.
4. Explicit documentation of the designed process itself, i.e., what are its components and their characteristics, how are they interconnected, what are their functional and performance characteristics, what are the critical design variables and the intrinsic tradeoffs.
5. Easy verification and modification of the resulting design. Having an explicit documentation of the intermediate design tasks, generated alternatives, rationale behind various design decisions, assumptions, conjectures, and simplifications, one can replay the design scenario and easily verify the validity of the derived processing scheme, or modify its design premises for further improvements.
6. The mechanized model of a design methodology offers an excellent depository for the organization of new empirical knowledge and/ or the systematic incorporation of new theoretical results and analytic tools. Such inclusion of new knowledge will progressively increase the automation of the process synthesis procedure itself.

1. Modeling the Process of Design

How do you construct a software system, which emulates a specific methodology for the synthesis of processing schemes? Figure 2 shows a generic, conceptual model of the design methodology. It is composed of three distinct components; a *Planner*, a *Scheduler*, and a *Designer*.

a. Planner. Defines a top-level milestones through which the state of the processing scheme is expected to pass. For the synthesis of processing schemes, using the Douglas hierarchical approach, these milestones are (1)

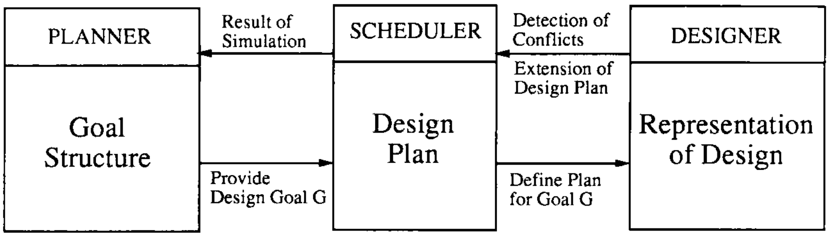


FIG. 2. A generic model for the design process.

design of the multiplant complex configuration, (2) design of the input/output structure for each plant, (3) design of the recycle structure, (4) design of the generalized separation structure, (5) synthesis of the separation subsystems (vapor, liquid, solid), (6) synthesis of heat exchanger networks, and (7) integration of processing subsystems.

b. Scheduler. Determines the sequence of design steps to be taken as one attempts to advance the current state of a processing scheme to the next milestone, defined by the Planner. It embodies a theory of how design goals and associated tasks are created, prioritized, and decomposed; how they interact, and how they are satisfied. As a result, it offers a fairly detailed design plan, by identifying all the requisite engineering design tasks. In subsequent sections, we will see that the *Scheduler* is represented by a tree of design tasks, which emulate the Douglas hierarchical design methodology (e.g., see Figs. 7–10, 12, 13).

c. Designer. It maintains the representation of the processing scheme being synthesized and other domain-specific knowledge. Thus, given a design step (from *Scheduler*), the *Designer* knows what must be done to execute the design step, e.g., reason with a specific set of rules, execute a design algorithm, carry out an optimization procedure, etc. It also updates the state of the evolving flowsheet, detects conflicts, and using domain-specific data, prescribes modifications to the design plan, which are communicated back to the *Scheduler*.

2. High-Level, Design-Oriented Languages

The conceptual model of the design process, discussed above (see also Fig. 2), should be turned into a logically well-defined computational process for the computer. There are high-level programming languages, such as FORTRAN, C, LISP, which can bridge the gap between the engineering model of conceptual design of Fig. 2, and its computer-based counterpart. However, as it has been pointed out by Newell (1982) and Chandrasekaran (1986), this gap is too wide for these languages to bridge in an expressive and efficient manner. Process designers cannot describe their design activities in their own terms, using FORTRAN, C, or LISP. Since the structure of language defines the boundaries of thought and expression, it is clear that the computer-based representation of a design methodology must rely on higher-level linguistic constructs, which allow us to do the following: (1) to represent the evolving structures of processing schemes in significant detail, including all the semantic relationships between the various representational components (processing units, streams, materials, modeling relationships, variables, decision-making procedures, numerical algorithms, etc.) and (2) to represent the design methodology itself, i.e., the intermediate design milestones, the goal structures taking the process from one milestone to the next, the procedures supporting various design tasks, etc.

3. Object-Oriented Representation of Knowledge

As a conceptual process design is a knowledge-intensive activity, its computer-based automation leads to a very large and complex computer program, which represents a very broad range of data models and procedural methodologies. Writing a large computer program is neither a trivial task, nor a straightforward exercise when all the design activities are already known.

The recent advances in object-oriented programming technology make this complex task much simpler by introducing the concepts of the *encapsulation* (or information hiding), *inheritance*, and *polymorphism*. In an object-oriented environment, the structure of the program can be designed in such a way that it represent a one-to-one mapping of the conceptual design methodology itself. For each object in engineering design methodology, a corresponding computational object can be constructed. For each engineering task, a symbolic operation possibly containing numerical computations in the computational model can be defined. With the use of this strategy, extending the model to accommodate new objects or new actions requires no strategic changes to the structure of the program itself. It only needs the addition of the new symbolic analogs of those objects or actions.

Many currently available CAD programs do not fully implement this concept. This is in part due to the unavailability of this technology when those systems were developed, and in part due to the lack of compatibility between the old codes and these new technologies. As a result, it is not a trivial task to modify the existing models of the CAD systems, or add a new model to the design system.

4. *Human–Computer Interface*

Given that the design process is such a complex process, the human–computer interface is very important for the user, in order to monitor the progress of the evolving design, provide decisions, and guide the direction of design in a user-friendly manner. It is not surprising that many computer-aided simulation environments, such as ASPEN Plus, PRO/II, DESIGN II, and HYSIM, have incorporated advanced graphic user interfaces into their environment. However, to help the process designer control the complexity of the design process, these user interfaces should be designed in a manner consistent to the model of the design process itself. The goal is to make the design process explicit to the user and allow the user to monitor the design process and communicate his/her ideas through user interfaces. When the user needs to make a design decision, the system should provide him or her with a context for the design decision. This context should have minimal information, yet still provide the key information necessary for the decisionmaking. After the user makes a design decision, the design decision should be recorded and made available to the user. The user should be able to see the effect of the design decisions on the process flowsheet, request and receive explanations, and solicit advice for the evolution of design. When the user interfaces satisfy these requirements, the user can take full control of the design process, while large segments of the design process are automated by the computer.

II. Hierarchical Approach to the Synthesis of Chemical Processing Schemes: A Computational Model of the Engineering Methodology

Douglas' hierarchical design approach (Douglas, 1985, 1988) was chosen to be the model of the engineering methodology, whose computer-based automation we seek to achieve. This approach has been used for the design of single-product continuous processes with single or multi-step reactions (Douglas, 1985, 1988, 1989), solid processes (Rossiter and Douglas, 1986a, b; Rajagopal *et al.*, 1992), polymer processes (McKenna and Malone, 1990), pharmaceuticals, and specialty chemicals

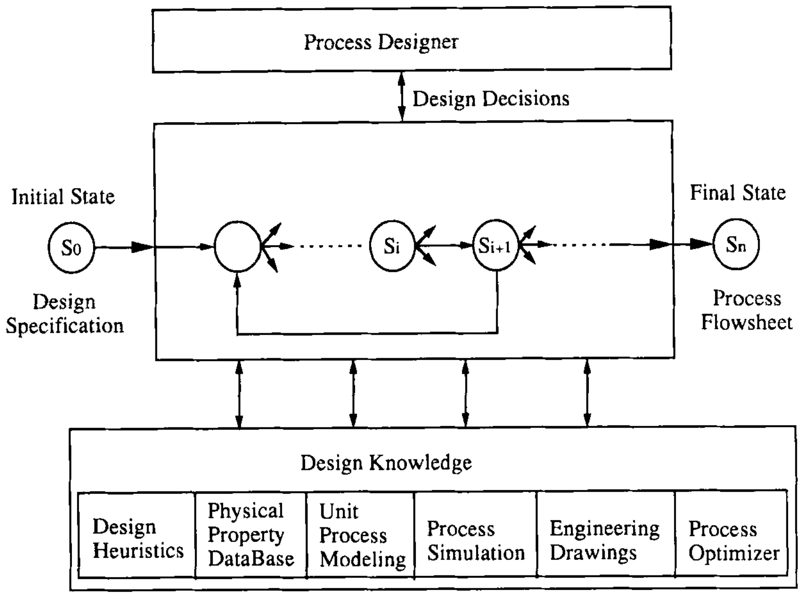


FIG. 3. Schematic diagram of the generic design process.

(Stephanopoulos *et al.*, 1994). It provides a well-defined structure of how the synthesis of conceptual processing schemes proceeds from very abstract, vague, and incomplete to fairly detailed, exact, and quite complete flowsheets. However, when it was developed, it was not intended to be a computational model for the design process itself. As a result, it lacks the requisite formalism and representational exactness needed for the construction of a computational process. In this section, we will provide a formalized restatement of the hierarchical procedure, in an effort to cast it more closely to a computational process for its computer-based automation.

A. HIERARCHICAL PLANNING OF THE PROCESS DESIGN EVOLUTION

It is common to characterize a design history as a series of transformations from specification to final design (Fig. 3). However, every design methodology does not handle the gap between "specifications" and "final design" in one sweeping structure of design steps. Instead, it identifies the abstract milestones through which the design is expected to go. In the case of the synthesis of processing schemes, the Douglas methodology has identified the following sequence of intermediate design milestones:

Beginning. Collect project specifications, e.g., chemistry to be used, feedstocks available, products to be produced, desired production specifications (amounts, purity), and economic constraints not to be violated (e.g., required capital $\leq a$, a product unit cost $\leq b$, return on investment $\geq c$, where a, b, and c are available bounding values).

Milestone 1. Synthesize and evaluate *plant complex structure*, i.e., identify the number of abstract "plants" each of which is formalized around a set of reactions, which take place under the same conditions (Fig. 4a).

Milestone 2. Design the *input/output structure* for each "plant," i.e., identify the set of *distinct* input and output streams associated with each "plant" of the overall structure (Fig. 4b).

Milestone 3. Select the *recycle structure* connecting the generalized abstract reaction and separation sections (Fig. 4c). During this phase of the design, the essential design alternatives of the generalized reaction section are also identified, i.e., the types of feasible reactors and their interconnections.

Milestone 4. Select the *generalized separation structure*, i.e., the elements of all-encompassing structure, shown in Fig. 4d, which are needed for the specific output from the reaction section of a given "plant."

Milestone 5. Synthesize the structure of the specific separation subsystems, e.g., vapor recovery, gas separation, liquid separation, or/and solid separation, which are relevant to the generalized separation structure of a given "plant."

Milestone 6. Integrate the process flowsheets, identified for each "plant" of the plant complex structure (Fig. 4a), and carry out consolidation of common processing tasks. The resulting design represents a flowsheet for the whole plant.

Milestone 7. Synthesize energy management systems for the consolidated flowsheet, derived at the last step.

End. Carry out sensitivity analyses of the process flowsheet's economic measures and revisit the critical design decisions, thus giving rise to improved designs.

It is clear from the above discussion that the Douglas approach to the synthesis of conceptual process flowsheets is based on the idea of *abstract refinement* (Mitchell *et al.*, 1981; Stephanopoulos, 1989), which is shown in Fig. 5. Since top-down abstract refinement does not handle goal coupling

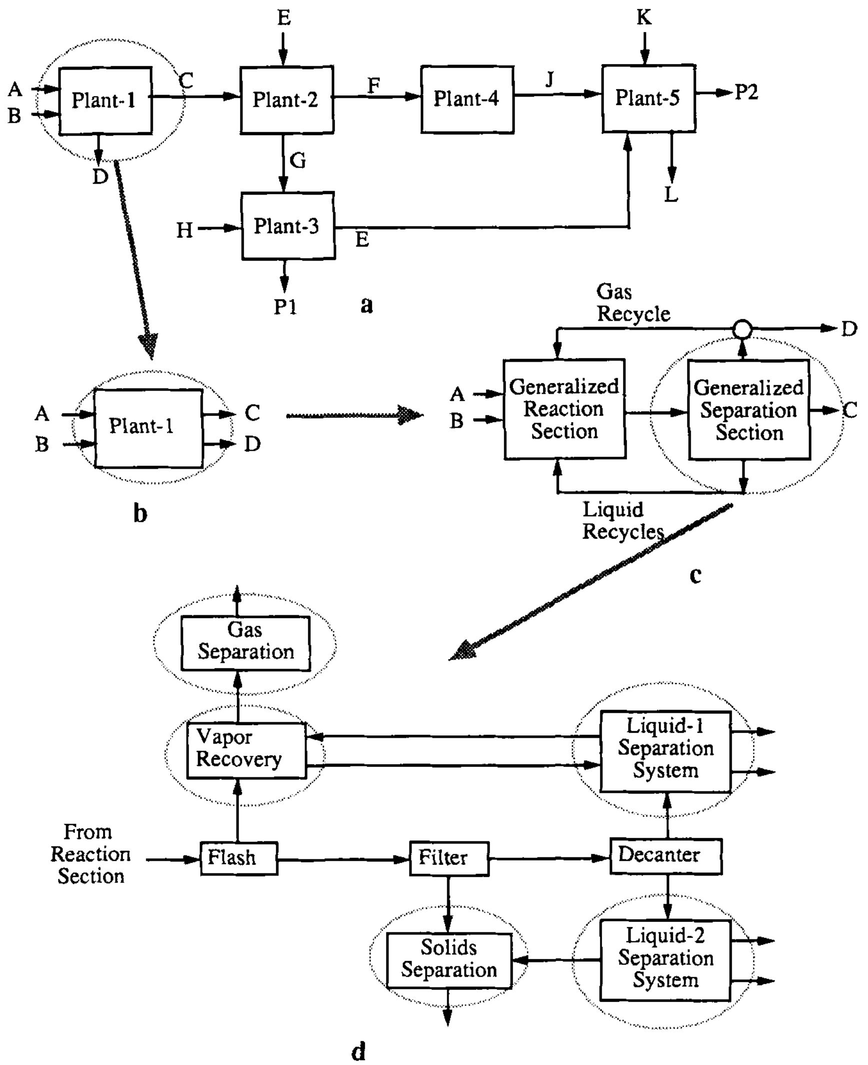


FIG. 4. Hierarchical evolution of conceptual processing schemes.

well, the approach must use, and in fact it does, *constraint propagation* to achieve consistency between the different parts of the overall processing scheme, as it will be discussed in a subsequent paragraph. However, for the time being, it is clear that Fig. 5 outlines the hierarchical planning stages for the conversion of specifications into one or more final process flowsheets.

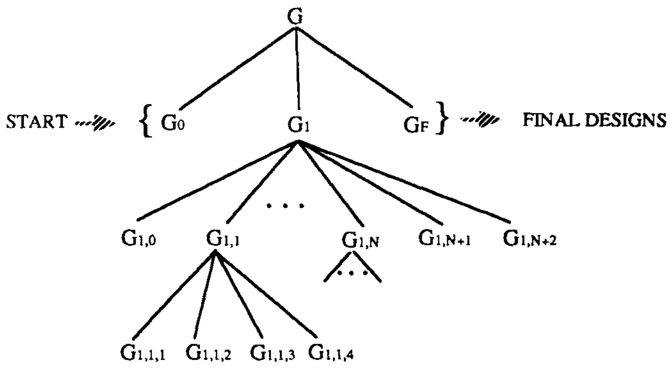


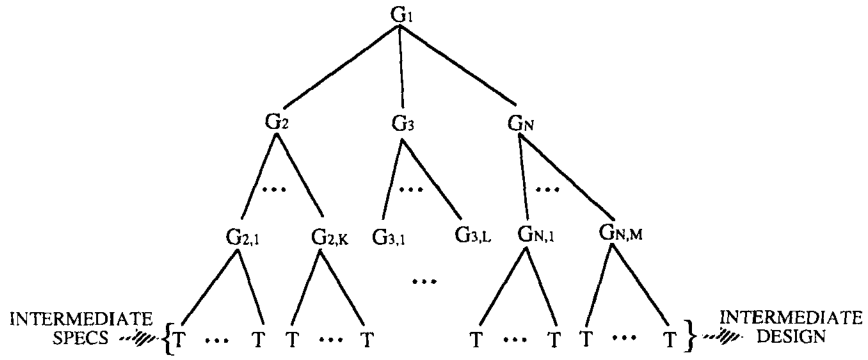
FIG. 5. Tree of goal structures in abstract refinement—(G—design the conceptual processing scheme for a multistep reaction plant; G_0—initialize the project's specifications; G_1—synthesize the process flowsheet for the whole-plant designs; G_F—evaluate process flowsheet and create improved designs; $G_{1,0}$—synthesize the plant complex structure; $G_{1,i}$—synthesize the flowsheet for plant-i; $G_{1,i,1}$—design the input/output structure for plant-i; $G_{1,i,2}$—design the recycle structure for plant-i; $G_{1,i,3}$—design the generalized separation structure for plant-i; $G_{1,i,4}$—design the liquid separation subsystem for plant-i; $G_{1,N+1}$—consolidate the flowsheets of all plant-i, $i = 1,2,\ldots,$into one; $G_{1,N+2}$—synthesize energy management systems for consolidated process).

B. Goal Structures: Bridging the Gap between Design Milestones

Moving the synthesis of process flowsheets from one milestone to the next entails a series of design steps with progressively increasing amount of details. These design steps need, nevertheless, to be articulated very explicitly and modeled by specific entities, before they can be automated by a computational procedure. In this section, we will focus our attention on the articulation of design tasks or steps, limiting our view to the synthesis of processes with a single fluid product with no solids present anywhere in the process.

1. *Goal Structures and the Transformational Model*

While the model of abstract refinement is quite satisfactory for the hierarchical planning of the process synthesis methodology, it is quite cumbersome and possibly incorrect as a model to describe the intricate web of design actions from one milestone to the next. Instead, the *transformational model* (Balzer *et al.*, 1976; Scherlis and Scott, 1983; Stephanopoulos, 1989) provides a more appropriate vehicle. This model converts specifications into designs through a series of correctness-



G: Goal T: Transformation to achieve corresponding goal

FIG. 6. Structure of design tasks in the transformational model.

preserving transformations from one complete description to another (see Fig. 6). Thus, a single transformation may operate on several components of an evolving flowsheet at once. It is more general than the abstract refinement, which is limited to steps that replace a single component with a more detailed description of it. However, the generality comes at the potential cost of increased complexity, since a transformation on a complete description must deal with more information and must tackle all design decisions simultaneously. Nevertheless, since a transformation sequence can be viewed as an executable program for implementing the desired specifications, it is in agreement with the nature of the computational tools available in process engineering, i.e., branch-and-bound algorithms to solve mixed-integer constraint- or optimization-based design problems. Let us see now how we can model the transformation of one design milestone to the next, using Douglas' methodology for the synthesis of conceptual processing schemes. In doing so, in the subsequent paragraphs, we will generate the goal structures and requisite tasks following the general transformational model shown in Fig. 6, which can be easily converted into a computational process.

a. Goal-Structure 1: Initialize Project Definition. Looking back to Fig. 5, we notice that goal G_0, i.e., initialize the project's specifications, is the first to be accomplished. Figure 7 shows the refinement of G_0 into a series of data-acquisition subgoals, all of which are satisfied by user actions. Table I provides an example of project specifications at the outset of the process synthesis for a plant for the hydrodealkylation of toluence (Douglas, 1988).

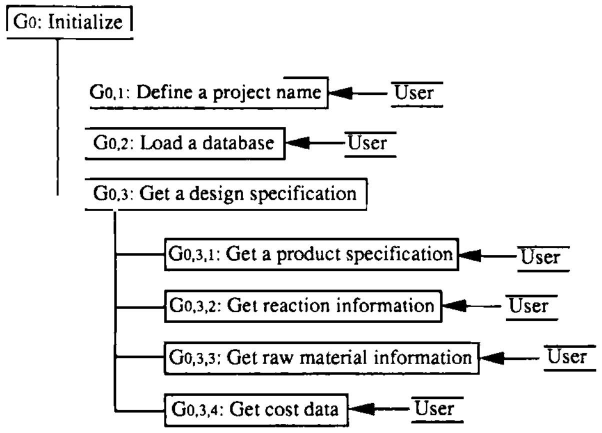

G0: Initialize
G0,1: Define a project name
User
G0,2: Load a database
User
G0,3: Get a design specification
G0,3,1: Get a product specification
User
G0,3,2: Get reaction information
User
G0,3,3: Get raw material information
User
G0,3,4: Get cost data
User

FIG. 7. Goal structure for the goal G_0, "Initialize the project specification."

b. Goal-Structure 2: Specify the Plant Complex Structure. For multistep reaction schemes, the synthesis of the process flowsheet starts by identifying the plant complex structure, i.e., achieving goal $G_{1,0}$ in Fig. 5. Figure 8 presents the transformational model of the goal structure, which identifies the individual "plants" defined around a set of reactions. From the design specification, the methodology can identify the number of simple plants. The reactions that take place at the same temperature, pressure, and catalysts are allocated to the same plant. The input and output streams of

TABLE I

DESIGN SPECIFICATIONS FOR HYDRODEALKYLATION (HDA) OF TOLUENE TO PRODUCE BENZENE[a]

Product specification
Product: benzene
Production rate: 265 lb-mol/h
Product purity: 0.9997 mole fraction
Reaction information
Toluene + $H_2 \rightarrow$ benzene + CH_4 at gas phase, 500 psia, 1150°F
Extent of reaction = prod/[1 − 0.0036/(1 − X)$^{1.5}$]
2 Benzene $\rightleftharpoons$ diphenyl + H_2 at gas phase, 500 psia, 1150°F
Extent of reaction = prod[0.0036/(1 − X)$^{1.5}$)/2(1 − 0.0036/(1 − X)$^{1.5}$]
Raw material information
Pure toluene at 15 psia, gas phase
H_2: 95%, CH_4: 5% at 550 psia, 100°F, gas phase

[a]Key: prod represents the molar flowrate of the product; X represents the conversion of the primary reaction; psia = pounds per square inch absolute.

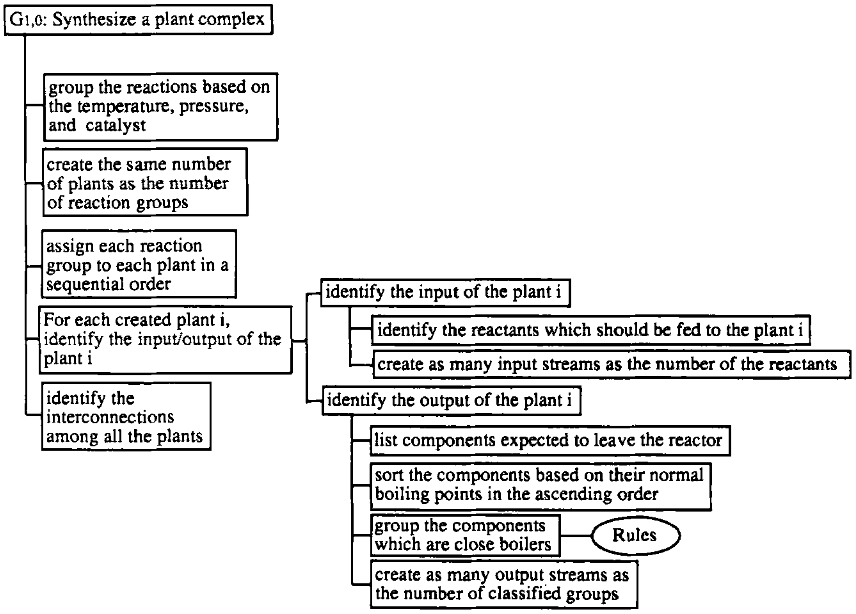

G1,0: Synthesize a plant complex
group the reactions based on the temperature, pressure, and catalyst
create the same number of plants as the number of reaction groups
assign each reaction group to each plant in a sequential order
For each created plant i, identify the input/output of the plant i
identify the interconnections among all the plants
identify the input of the plant i
identify the reactants which should be fed to the plant i
create as many input streams as the number of the reactants
identify the output of the plant i
list components expected to leave the reactor
sort the components based on their normal boiling points in the ascending order
group the components which are close boilers
Rules
create as many output streams as the number of classified groups

FIG. 8. Goal structure for the goal $G_{1,0}$, "Synthesize the plant complex structure."

each plant are identified. The rules (Douglas, 1988) for deciding the destination code are used to decide the number of output streams and the destinations of output streams. Once input and output streams of each plant have been identified, we can establish the interconnections among the plants using stream matching.

c. Goal-Structure 3: Design the Input-Output Structure. For each "plant-i" identified above, the hierarchical design of a process starts by selecting the structure of input and output streams. The design tasks at this stage are depicted in the transformational model of the goal structure shown in Fig. 9 and described below.

(i) *Synthesis.* When there are impurities in the feed streams, the methodology needs to make a decision for each feed stream about whether it needs feed pretreatment before the plant system.

(ii) *Analysis.* The design variable is a conversion for the primary reaction. If there are multiple reactions, then extents of reactions should be provided as functions of a conversion for the primary reaction, product

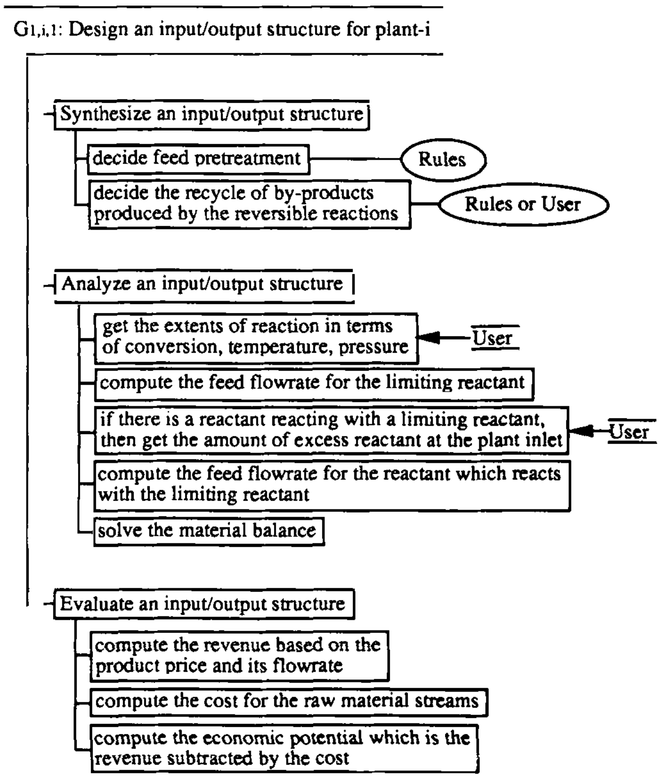

G1,i,1: Design an input/output structure for plant-i
Synthesize an input/output structure
decide feed pretreatment
Rules
decide the recycle of by-products produced by the reversible reactions
Rules or User
Analyze an input/output structure
get the extents of reaction in terms of conversion, temperature, pressure
User
compute the feed flowrate for the limiting reactant
if there is a reactant reacting with a limiting reactant, then get the amount of excess reactant at the plant inlet
User
compute the feed flowrate for the reactant which reacts with the limiting reactant
solve the material balance
Evaluate an input/output structure
compute the revenue based on the product price and its flowrate
compute the cost for the raw material streams
compute the economic potential which is the revenue subtracted by the cost

FIG. 9. Goal structure for the goal $G_{1,i,1}$, "Design an input/output structure for plant-i."

flowrate, temperature, and pressure. The extents of reactions represent the selectivity. The excess reactant ratio at the plant inlet is another design variable. It represents the ratio between the amount of the key reactant and that of the limiting reactant. A composition in the gaseous outlet stream can be substituted for the excess reactant ratio at the plant inlet.

(iii) *Evaluation.* Economic potential is computed by subtracting the costs of raw materials and feed pretreatment systems from the revenue from the product and the byproducts.

(iv) *Refinement.* The plant system identified at the input/output structure level is refined at the recycle structure level. The plant system is used as a boundary system at the recycle structure level.

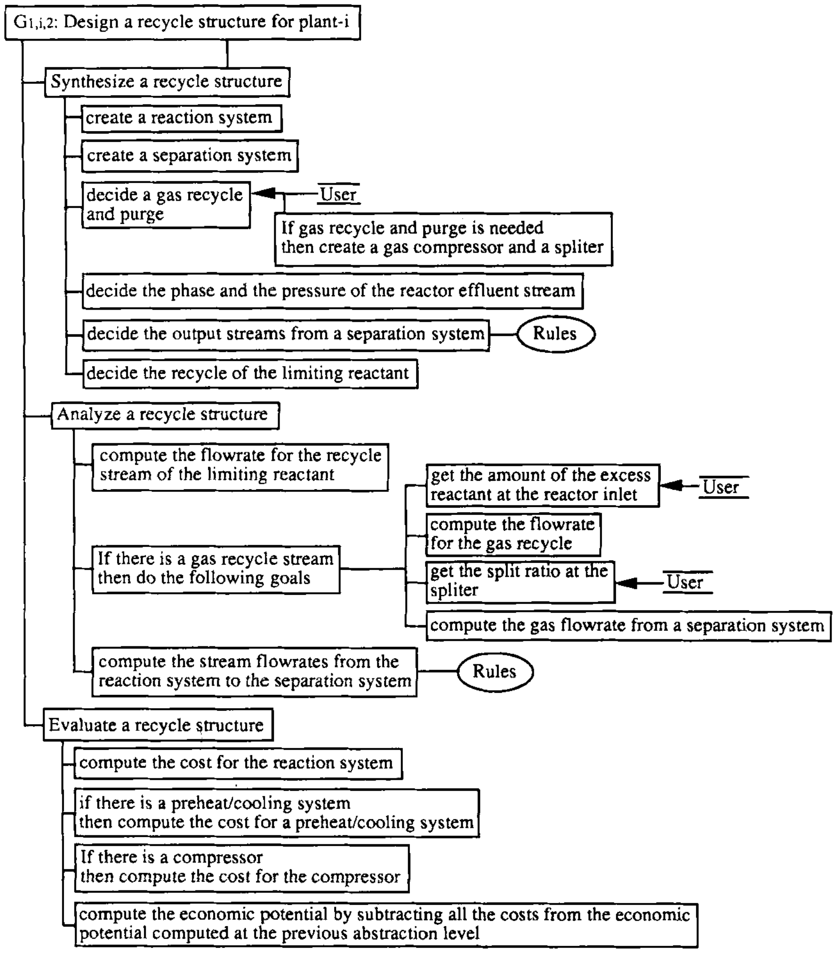

G1,i,2: Design a recycle structure for plant-i
Synthesize a recycle structure
create a reaction system
create a separation system
decide a gas recycle and purge
User
If gas recycle and purge is needed then create a gas compressor and a spliter
decide the phase and the pressure of the reactor effluent stream
decide the output streams from a separation system
Rules
decide the recycle of the limiting reactant
Analyze a recycle structure
compute the flowrate for the recycle stream of the limiting reactant
get the amount of the excess reactant at the reactor inlet
User
compute the flowrate for the gas recycle
If there is a gas recycle stream then do the following goals
get the split ratio at the spliter
User
compute the gas flowrate from a separation system
compute the stream flowrates from the reaction system to the separation system
Rules
Evaluate a recycle structure
compute the cost for the reaction system
if there is a preheat/cooling system then compute the cost for a preheat/cooling system
If there is a compressor then compute the cost for the compressor
compute the economic potential by subtracting all the costs from the economic potential computed at the previous abstraction level

FIG. 10. Goal structure for the goal $G_{1,i,2}$, "Design a recycle structure for plant-i."

d. Goal-Structure 4: Design the Recycle Structure. The input/ output structure of "plant-i" is refined into a more structured network of two sections; the generalized reaction and separation subsystems (e.g., see Fig. 4c). Figure 10 shows the goal structure, depicting the tasks needed for satisfying the top-goal, $G_{1,i,2}$, following again the transformational model in Fig. 6. The design tasks are discussed below.

(i) *Synthesis.* It is assumed that the recycle structure consists of a reaction section and a separation section. It is also assumed that 99% recovery is the same as 100% recycle. Thus, the unreacted limiting reactant will be recycled completely. The user needs to make a design decision about either just purging the gas stream that has reactants, or recycling a part of the gas stream and purging the remaining gas stream. As a result of this decision, a gas compressor and a spliter may be added to the flowsheet. When the plant system has a secondary reversible reaction, the user has to make a design decision about whether it is desirable to shift the equilibrium by recycling the byproduct.

(ii) *Analysis.* As the number of design alternatives is small, the material balance can be solved based on the structural decisions done at the synthesis phase. The ratio between the amount of the limiting reactant and that of the key reactant at the reactor inlet is needed as a parametric decision variable. This design variable indicates the excess amount of key reactant at the reactor inlet.

(iii) *Evaluation.* Economic potential is computed by subtracting the costs for the reaction system and gas compressor from the economic potential computed at the input/output structure level. To compute the costs for those units, several assumptions, such as the reactor type and the kinetic model, should be made. More detailed algorithms and example applications are available in Douglas' design text (Douglas, 1988).

(iv) *Refinement.* The separation system identified at the recycle structure level is refined at the general separation structure level. The separation system is used as a boundary system at the general separation structure level.

e. Goal-Structure 5: The General Separation Structure. The refinement of the generalized separation section leads to the superstructure of Fig. 4d, which accounts for all possible configurations of abstract separation subsystems. We will limit our attention to processes with fluids and single liquid phase only, thus eliminating the need for a filter, decanter and the associated subsystems of Fig. 4d. The resulting feasible options are shown in Fig. 11. Figure 12 shows the goal structure of the transformational model, with the corresponding tasks organized as follows.

(i) *Synthesis.* It is assumed that the general separation structure will be one of the three structures shown in Fig. 11. A structure is chosen on the basis of the phase of the reactor outlet stream. When the phase of the reactor effluent stream is not liquid, the user needs to make a design decision about whether it needs a vapor recovery system. If it does, the user should decide the location of the vapor recovery system. The location will be one of "on the recycle stream," "on the spliter outlet stream," or

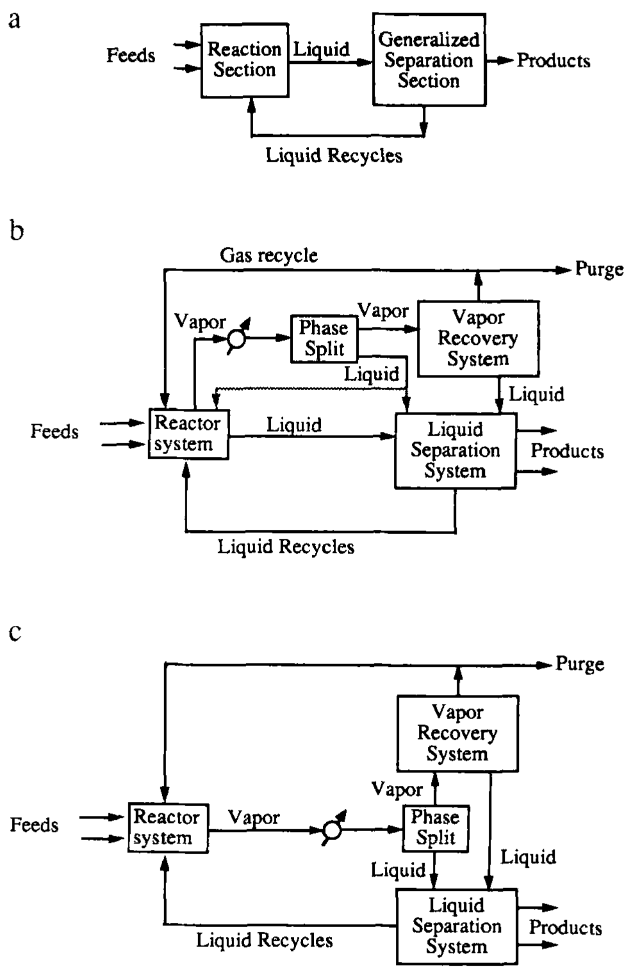


FIG. 11. Three possible general separation structures when reactor exit is (a) liquid (b) vapor, and liquid (c) vapor only [Reproduced from Hierarchical Decision Procedure for Process Synthesis, J. M. Douglas, *AIChE J.*, **31**, 353 (1985), by permission].

"on the purge stream." To help the user make this design decision, the system makes the flash calculation for the spliter and shows the flowrate and compositions of the vapor-phase outlet stream from the spliter. If the user decides to locate the vapor recovery system on the recycle stream to the reaction system, the control automatically goes back to the recycle structure level and locates the vapor recovery system on the recycle stream, and then the design procedure restarts at the general separation structure level.

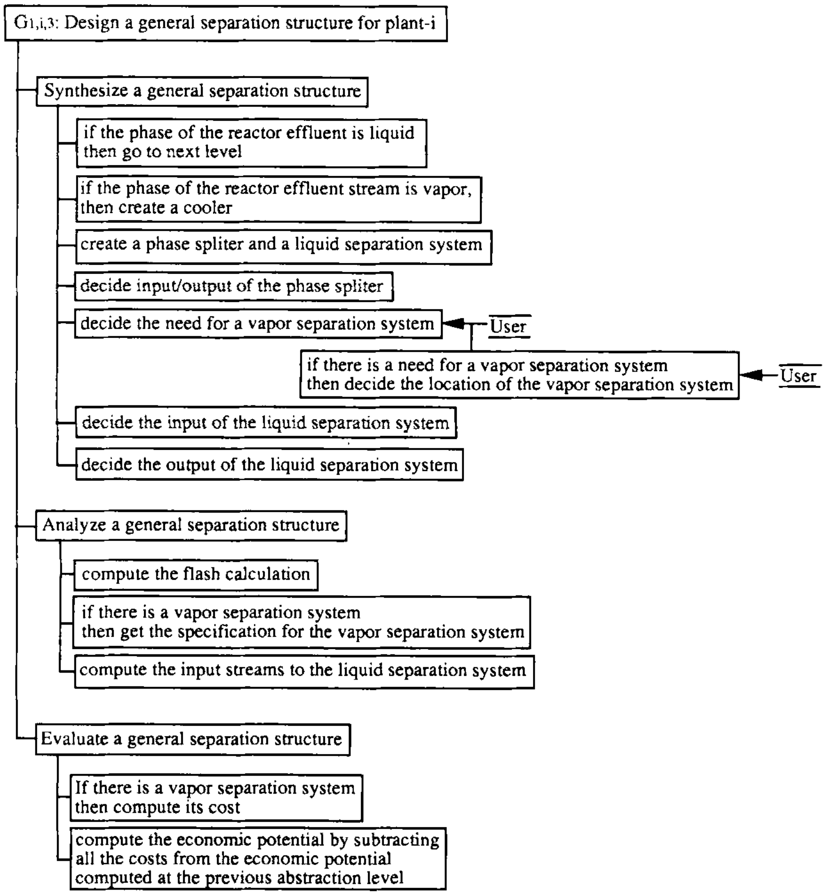

G1,i,3: Design a general separation structure for plant-i
Synthesize a general separation structure
if the phase of the reactor effluent is liquid then go to next level
if the phase of the reactor effluent stream is vapor, then create a cooler
create a phase spliter and a liquid separation system
decide input/output of the phase spliter
decide the need for a vapor separation system
User
if there is a need for a vapor separation system then decide the location of the vapor separation system
User
decide the input of the liquid separation system
decide the output of the liquid separation system
Analyze a general separation structure
compute the flash calculation
if there is a vapor separation system then get the specification for the vapor separation system
compute the input streams to the liquid separation system
Evaluate a general separation structure
If there is a vapor separation system then compute its cost
compute the economic potential by subtracting all the costs from the economic potential computed at the previous abstraction level

FIG. 12. Goal structure for the goal $G_{1,i,3}$, "Design a general separation structure for plant-i."

(ii) *Analysis.* If a vapor separation system is in a flowsheet, the user should give a specification for the vapor separation system, e.g., the existing components and their compositions at the liquid-phase outlet stream from the vapor separation system. As there are no more degrees of freedom, all the material and energy balance calculations can be done without any interaction with the user.

(iii) *Evaluation.* Economic potential is computed by subtracting the cost of vapor recovery system from the economic potential computed at the previous level. To compute the cost for a vapor separation system, the

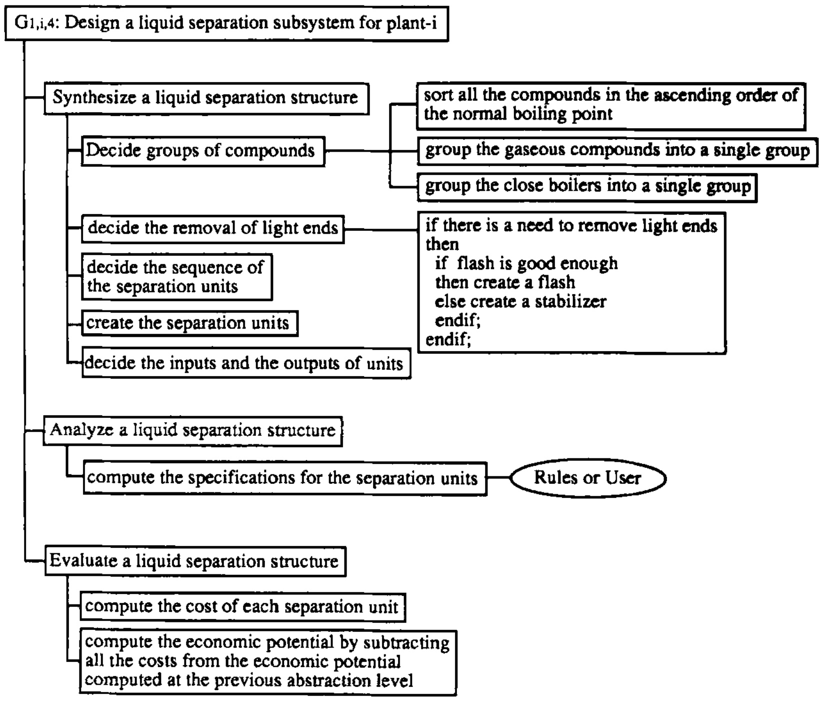

G1,i,4: Design a liquid separation subsystem for plant-i
Synthesize a liquid separation structure
Decide groups of compounds
sort all the compounds in the ascending order of the normal boiling point
group the gaseous compounds into a single group
group the close boilers into a single group
decide the removal of light ends
if there is a need to remove light ends then if flash is good enough then create a flash else create a stabilizer endif; endif;
decide the sequence of the separation units
create the separation units
decide the inputs and the outputs of units
Analyze a liquid separation structure
compute the specifications for the separation units
Rules or User
Evaluate a liquid separation structure
compute the cost of each separation unit
compute the economic potential by subtracting all the costs from the economic potential computed at the previous abstraction level

FIG. 13. Goal structure for the goal $G_{1,i,4}$, "Design a liquid separation subsystem for plant-i."

type of the system, e.g., absorption, condensation, should be specified. The algorithm for the absorber costing is available in Douglas' design text (Douglas, 1988).

(iv) *Refinement*. The liquid separation system identified at the general separation structure level is refined at the liquid separation structure level. The liquid separation system is used as a boundary system at the liquid separation structure level.

f. Goal-Structure 6: The Liquid Separation Subsystem. Figure 13 shows the goal structure and associated tasks during the design of the liquid separation subsystem.

(i) *Synthesis*. As the liquid separation system has only liquid feed streams, the only synthesis subproblem is that of the synthesis of the liquid separation sequence. If the input streams to the liquid separation system have light ends, then the methodology checks whether the light ends should be removed to satisfy the product specification. If light ends need to be removed, the methodology considers the flash as a means to remove the light ends. If the flash is not enough, then the design methodology decides to put a stabilizer column as a first column in the separation sequence. For the remaining components, the system decides the separation sequence. When the process has only three components to be separated, then, Eq. (1) is used to select either the direct or indirect sequence (Malone *et al.*, 1985). Whenever the left-hand side of Eq. (1) takes on a negative value, the indirect sequence is preferred.

$$\frac{\delta V}{F} = 1.2\left\{\left(\frac{x_B + x_C}{a_{BC}}\right)\left(\frac{x_A x_C}{1 + x_A x_C}\right) + \left(\frac{1}{a_{AC} - 1}\right)\left(\frac{x_C - f x_A + x_A x_C^2}{f(1 + x_A x_C)}\right) - \frac{a_{BC}(x_A + x_B)(f - 1)}{(a_{AC} - a_{BC})f}\right\} - x_A, \quad (1)$$

where a_{xy} is a relative volatility between component x and component y, x_A is a mole fraction of component A in the feed stream, and f is a correction factor, $f = 1 + 1/100x_B$.

When the process has more than three components, the selection is based on either heuristics (Douglas, 1988), or an implicit enumeration of the alternative sequences.

(ii) *Analysis.* It is assumed that in every separation unit, 99.5% of the light key is recovered in the overhead, and 99.5% of the heavy key in the bottom. It is also assumed that all the components lighter than the light key are taken overhead and that all components heavier than the heavy key leave with the bottom.

(iii) *Evaluation.* Economic potential is computed by subtracting all the costs of the distillation units from the economic potential computed at the previous abstraction level. To compute the cost of each separation unit, Fenske–Underwood–Gilland methods are used. To approximate the behavior of the distillation unit, a user can supply the average relative volatility for the column.

C. Design Principles of the Computational Model

Let us try to summarize the main principles (Han, 1994) on which a formalized computational model (outlined in the previous two sections) of the Douglas conceptual process design methodology was based (Douglas, 1988).

1. *Principle 1: Hierarchical Planning of the Design Methodology*

The overall design methodology was modeled as a hierarchical planning process. It starts with project specifications, goes through a predefined set of intermediate design milestones, and ends with a number of design alternatives.

2. *Principle 2: Successive Refinement of Specifications into Implementations*

The principle of hierarchical planning by itself cannot identify the design characteristics of the intermediate milestones. Successive refinement, on the other hand, does. Thus, at each stage of the hierarchical planning, we have a refinement of the specifications with more detailed description of the design characteristics of the next milestone. Figure 4 shows how the implementation specifications are successively refined from those of the input/output structure (first implementation), to recycle structure (second implementation), to generalized separation structure (third implementation), to separation subsystem structures (fourth implementation), etc. The generic *goal structure* of Fig. 5, based on the notion of *abstract refinement*, is the essential model that has been used to capture both the hierarchical planning and the successive refinement of the overall design methodology.

3. *Principle 3: Propagation of Constraints*

The top-down refinement of the abstract refinement model does not handle goal coupling well. Thus, the refinement of an abstract unit into a network of subunits, leads to design problems which could be solved independently; clearly, a gross violation of the overall system's intended functionality. Consequently, *constraint propagation*, to achieve consistency between the different parts of the design, is essential. Constraint propagation manifests itself through two distinct mechanisms; *induction of new constraints* and *propagation of constraint values*. Let us discuss two mechanisms in more detail since both appear during the synthesis of conceptual process design.

a. Propagation of Constraint Values. Consider the two abstractions of a process, shown in Fig. 14. At the input/output level (Fig. 14a), the input and output streams have flows as shown. At the recycle level (Fig. 14b), the input and output streams through the dashed boundary must have the same values, if the two abstractions are to be consistent. Thus, constraints on the flows of A, B, P, and (BP, A) have been propagated from one abstraction to the next.

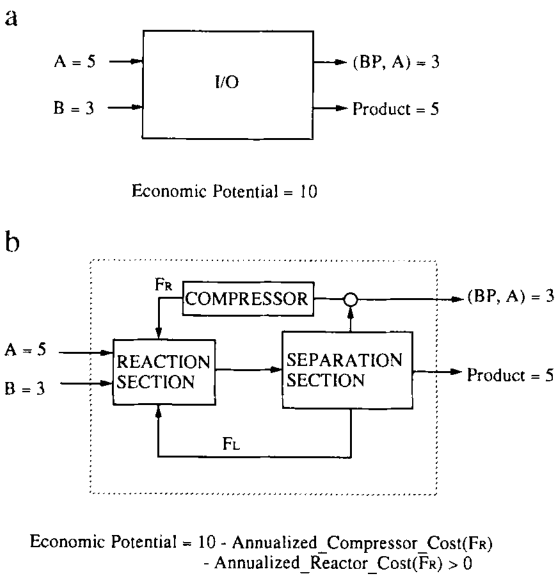


FIG. 14. Schematic used to indicate propagation of constraints from the input/output structure (a) to the recycle structure (b).

b. Induction of New Constraints. The design abstraction of Fig. 14b has revealed the presence of a compressor on the gas recycle stream. Since the economic potential must remain positive, the following constraint on the value of F_R is induced:

$$\text{Economic potential} = 10 - \text{annualized\_compressor\_cost}(F_R) - \text{annualized\_reactor\_cost}(F_R) > 0$$

Thus, as design constraints are propagated from abstraction i, to abstraction $i + 1$, they also induce new constraints that must be satisfied by the implementation at level $i + 1$.

4. Principle 4: Hierarchical Decomposition with Coordination

Each refinement consists of two steps: a decomposition and a coordination. Whenever a system is decomposed into a set of subsystems, a specific knowledge about how the current system should be decomposed into a set of subsystems is required. This knowledge should come from the problem context. For example, at the recycle structure level, the knowledge that a plant system will be decomposed into a reaction system and a separation system, is specific only for the decomposition at the recycle structure level. At the coordination phase, the inputs and outputs of all process units and their interconnections are identified. A coordination step is necessary because the decomposition of a system into a set of subsystems introduces additional details to the design. It should be emphasized that the coordination is accomplished through the two constraint propagation mechanisms discussed above.

5. *Principle 5: Unified Transformational Design*

The model of abstract refinement is satisfactory only for the depiction of the intermediate design milestones. It is unacceptable as the model to describe the design steps at each abstraction level. Instead, we have relied on the transformational model represented by the generic goal structure of Fig. 6. Therein we notice that the generation of an intermediate design is accomplished through a series of tasks, which preserve the correctness (i.e., constraint satisfaction and optimality) of the derived designs. It is important to note that the generic goal structure of Fig. 6 is an abstract notation of the intended transformations. Thus, it does take different forms depending on the specifics of the design methodology employed. For example, the goal structures of Figs. 9, 10, 12, and 13, all of which are manifestations of the transformational model, *are not unique* and simply signify the methodological design steps employed by the Douglas approach. One can envision an implicit enumeration algorithm, where the various tasks correspond to the computational steps of the particular algorithm.

In the present work, we have opted for an explicit description of the intended design goals along with the associated design tasks, thus giving rise to explicit transformational models at each level of abstraction, as shown by the detailed tasks in Figs. 9, 10, 12, and 13.

Furthermore, it is important to underline that the model of the unified transformational design has an internal structure that is fairly generic, and in all likelihood would be present in any specific implementation, namely, the cycle of *synthesis*, *analysis*, *and evaluation*. As can be seen from the goal structures in Figs. 9, 10, 12, and 13, synthesis, analysis and evaluation are the three common pivotal goals around which the specific design implementations are expressed.

Synthesis is the first and represents the activity of generating structural designs. When the design specifications are given as an input, the structure is synthesized by heuristic reasoning, algorithmic procedures, or based on the interaction with the user. During the synthesis phase, all the process units and the process streams are identified. Thus, the design is given to the analysis phase.

In the second phase, analysis implies the setting up and solving material and energy balances for the synthesized structure. When the degrees of freedom for the structure exceed zero, or the structure is underdefined, optimization search algorithms or the user supply as many values as the degrees of freedom. These variables are selected by the design heuristics.

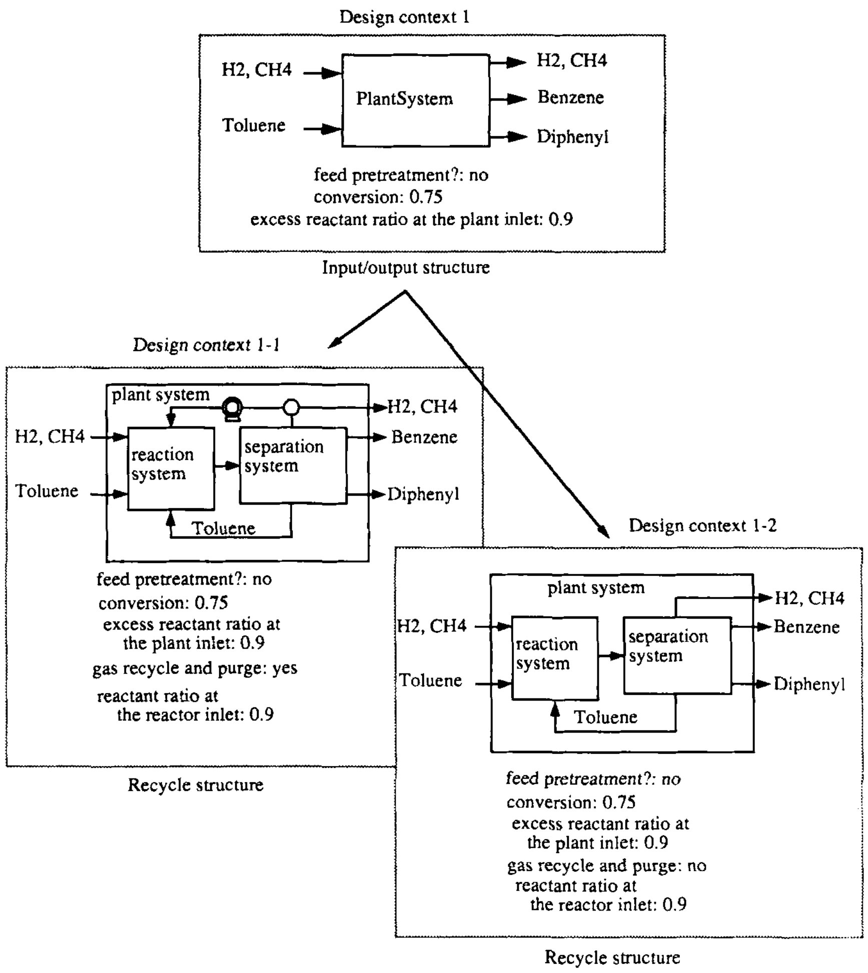


FIG. 15. Context-based design. This figure illustrates the scoping relationships among design contexts. The design decisions at the recycle structure are within the scope of the design decisions at the input/ output structure level.

After the analysis phase, all process units and streams have quantitative values (flowrate, compositions, etc.).

Evaluation is the third phase and represents the activity of evaluating the structure in terms of economic measures, economic potential (EP), defined as follows:

$$EP_i = \begin{cases} \text{revenue} - \text{cost}, & \text{at } i = 0, \\ EP_{i-1} - \text{cost} & \text{at } i > 0, \end{cases} \tag{2}$$

where i represents the abstraction level.

Clearly, when implicit enumeration techniques are used, the parametric and structural optimization carries out synthesis, analysis, and evaluation in an integrated, unbroken cycle of simultaneous computations.

6. *Principle 6: Context-Based Design*

The design at the current abstraction level is used as a specification that will be implemented at the next abstraction level and defines the scoping relationship between the abstraction levels. This relationship is important to understand the mechanism for managing alternative designs. On the basis of this scoping relationship, we can identify the hierarchical dependency relationships among designs at all the abstraction levels. Figure 15 illustrates a design context generated from such relationships. The design decision "feed pretreatment?" has been made at the input/ output structure. This design decision affects not only the input/ output structure, but also all the designs refined from this input/ output. For example, a local decision, "gas recycle and purge," shown in Fig. 15, is within the scope of the global decision, "feed pretreatment".

III. HDL: The Hierarchical Design Language

In the previous section, we discussed the structure of a conceptual model that can be used to represent a design methodology. But the value of this model rests with the effectiveness of the representation schemes that one employs to describe the declarative and procedural components of the model in a way that the computer can "understand." Thus, we are led to the need of defining a design-oriented language for the description of the computational process.

From the discussion in Section II, it is clear that a design-oriented language must address the following two needs:

1. Multifaceted modeling of the various states of the evolving process design.
2. Modeling of the procedural design tasks, as these are described by the goal structures.

HDL (Han, 1994) has been designed to meet the above two classes of modeling needs. It has been influenced by other previous work on process-design-related languages, such as MODEL.LA. (Stephanopoulos *et al.*, 1990a, b; see also first chapter in this volume), ConStruct (Johnston, 1991), SYDERELA (Kritikos, 1991), and ASCEND (Advanced System for Computations in Engineering Design) (Piela, 1989). The structure of HDL conforms with the design principles discussed in the previous section, and the object-oriented modeling, and human–computer interface requirements discussed in Section I. HDL provides a fairly simple and powerful framework of modeling classes, which the user can extend to develop a customized design environment. It should be emphasized that HDL is a formal framework for the development of the computational process that emulates the Douglas methodology for conceptual process design. In the following paragraphs, we will discuss its specific characteristics.

A. Multifaceted Modeling of the Process Design State

As the process flowsheet of a chemical plant is designed, it goes through a series of design states with variable details, with each design state being a snapshot description of the process flowsheet. Such a description encompasses the physical entities that are parts of the design. In addition, it includes the relationships among these variables. Thus, the declarative representation defines what a design or partial design is. Clearly the rationale, expressed in the first chapter in this volume, for a language such as MODEL.LA., finds a perfect example in the needs of HDL. Therefore, it is not surprising that the multi-faceted modeling capabilities of HDL, have drawn heavily from the structure of MODEL.LA.

1. Basic Modeling Elements

When we describe the chemical processes, a model should fully highlight the problem at hand, although not at the expense of excluding other possibly coexisting models. For example, models of chemical processes have modeling primitives corresponding to process units (e.g., reactors, pumps, pipes). Models of chemical processes also have primitives for aggregated units (e.g., reaction system, separation system, plant systems). So, although both design tasks are in the same domain, and both designs ultimately are composed of the vessels, pumps, pipes, etc., different modeling elements are used to highlight the important features of the design. Designs using the appropriate modeling elements convey the necessary design information with greater comprehension, because modeling primitives match intuitive description elements. These primitives can then be used to create an effective declarative representation.

To account for all these requirements, HDL provides a set of basic modeling elements drawn and extended from MODEL.LA (see first chapter in this volume, Stephanopoulos *et al.*, 1990a), which are used to represent the states of evolving process flowsheets at varying levels of abstraction. They also provide an efficient vehicle for the representation of alternative process designs with consistent referencing to common design elements of the alternative processing schemes. The ability to provide hierarchical representations and consistent versioning, allows HDL a truly multifaceted modeling of processing schemes.

Let us present HDL's basic modeling elements. For a pictorial depiction of their character, see Fig. 16. The first three modeling elements are sufficient to represent the "structure" of any processing system. The following two elements are used to represent complex systems which consist of networks of instances of `GenericUnit`, `Port`, and `Stream`.

a. Modeling-Element 1. `GenericUnit`. This is identical in character to the MODEL.LA.'s modeling element with the same name. It is used to represent an isolated system by defining the boundary that separates it from the surroundings. Thus, any system can be modeled as a `GenericUnit`, if the system has a clear boundary between itself and the environment. For example, a plant can be modeled as a `GenericUnit`. A reactor can also be modeled as a `GenericUnit`. To distinguish a plant from a reactor, the `GenericUnit` has two subclasses that have more specific properties. They are the `GenericSystem` and the `GenericProcessUnit`. The `GenericSystem` is the unit that can be decomposed into subsystems. For example, within the context of the hierarchical decision procedure, the plant system at the input/ output structure level is decomposed into a reaction system and a separation system at the recycle structure level. On the other hand, `GenericProcessUnit` is the unit that cannot be decomposed into subsystems without losing its physical identity. For example, when a reactor is decomposed into a vessel and a heating coil, a vessel does not have the function "reactor" any more.

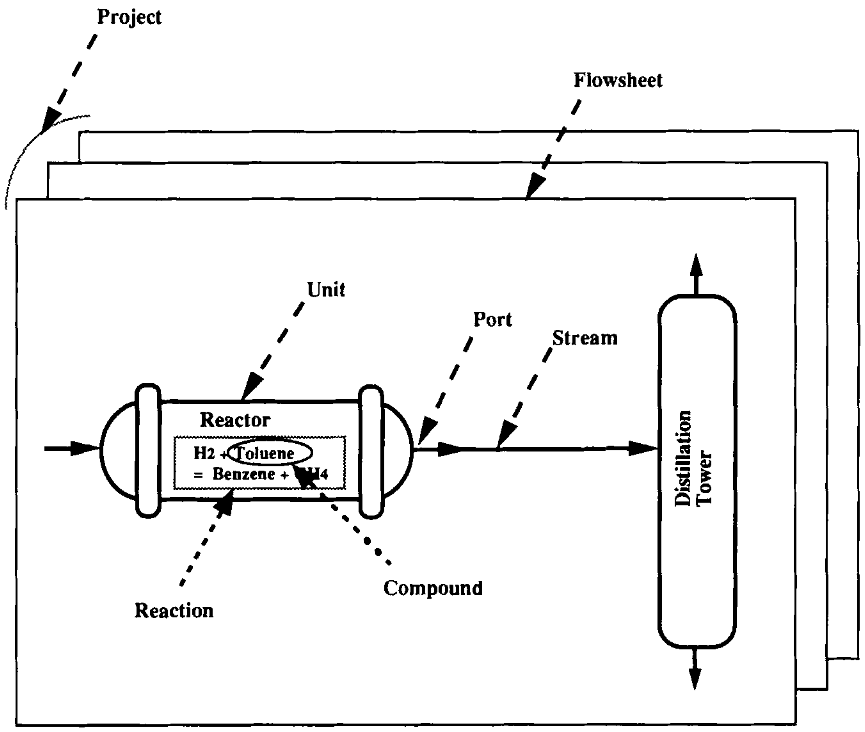


FIG. 16. Basic modeling elements in the HDL (hierarchical design language).

b. Modeling-Element 2: `Port`. As in MODEL.LA., instances of `Port` are used to define the boundaries of instances of `GenericUnit` and to provide the vehicles through which these instances transfer data to each other.

c. Modeling-Element 3: `Stream`. `Stream` is an abstraction that connects a port of an instance of `GenericUnit` to another port of another instance of `GenericUnit`. `Streams` know the `Ports` they are attached to, and symmetrically, each `Port` knows the stream to which it is attached. In logically consistent connections of `GenericUnits`, the flow through a stream from one port is connected to the another port of the same type.

d. Modeling-Element 4: `Flowsheet`. `Flowsheet` is an abstraction for a process flowsheet. It consists of process units and process streams. Except when the `Flowsheet` represents the design state at the highest abstraction level (least detail level), the `Flowsheet` has a boundary system which defines the design scope. The boundary system is transferred from the previous abstraction level. The user can monitor the design process through the `Flowsheet` and get the information which he/she wants to know from the `Flowsheet`. Each design alternative at each abstraction level is represented as an instance of `Flowsheet`.

e. Modeling-Element 5: `Project`. `Project` is an abstraction for a design project. An instance of `Project` keeps all the process flowsheets generated during the design project and handles the requests from a user, e.g., shows the `Flowsheet` at the recycle structure level. When a process alternative is generated, the alternative is also managed by the `Project`. Every instance of `Project` contains a tree-like data structure, in which it stores the state of the evolving `Flowsheet`. Specifically, as shown in Fig. 20, the root of the tree is the set of design specifications (empty `Flowsheet`), the first-level nodes represent alternative input-output structures, the second-level nodes the emanating recycle structures, etc. Clearly, the link between two connected nodes represents the design decision(s) generating one `Flowsheet` from the other.

f. Modeling-Element 6: `GenericVariable`. `GenericVariable` is an abstraction for a variable and is used to construct mathematical models for `GenericUnits`. It encapsulates the following information about a variable: variable name, variable value, possible range of values, and units. This additional information can be used to post constraints. These constraints are used to check whether the variable has a physically feasible value, e.g., a molar composition should be within the range from 0 to 1. The variable name and variable value type can be used to transfer information about the variable to the different environment, e.g. transfer the variable from HDL to Nexpert, which is a shell for the construction of expert systems, or copying a value from Nexpert to HDL.

g. Modeling-Element 7: `Compound`. `Compound` is an abstraction for a chemical compound. It has various elementary physical properties and the operations to access them from a database or to estimate derived properties through models and correlations. The elementary properties include normal boiling point, molecular weight, and critical temperature. The derived properties include the heat capacity at the given temperature or the vapor pressure at a given temperature.

h. Modeling-Element 8: `Reaction`. `Reaction` is an abstraction for a single reaction. It has information about the reactants, products and their stoichiometric coefficients, reaction temperature, reaction pressure, etc. It

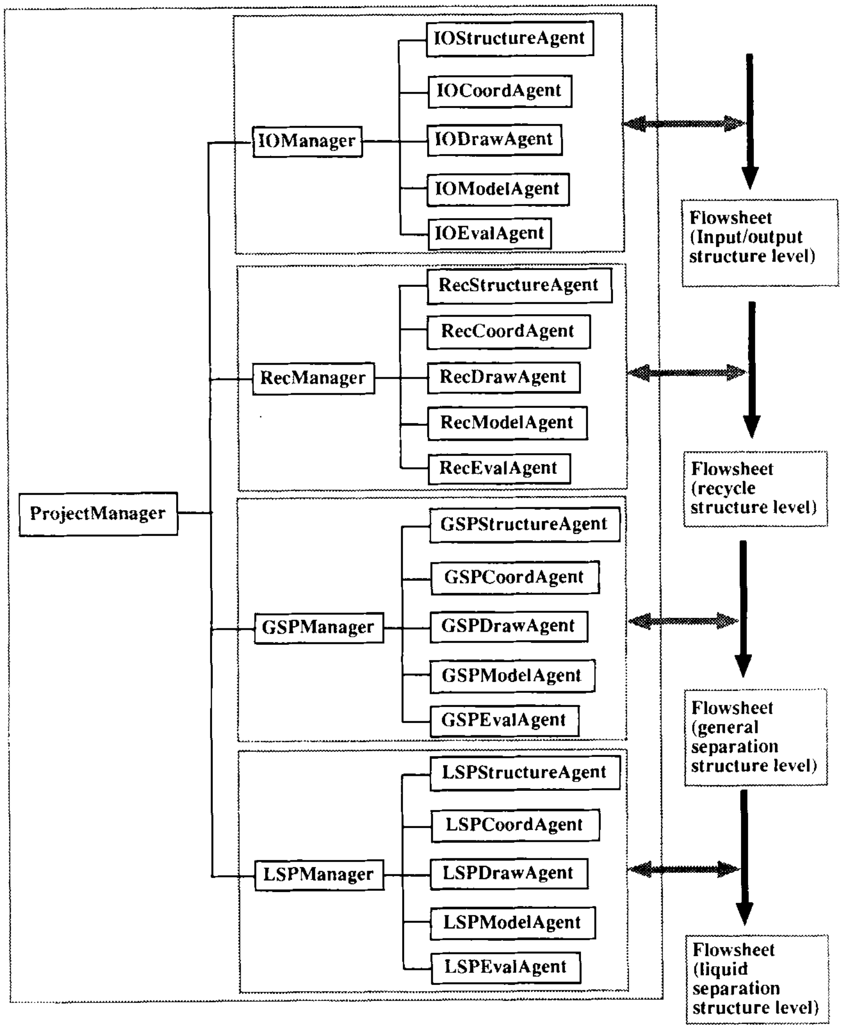


FIG. 17. Object model for design managers and design agents.

also has operations to estimate the properties such as the heat of reaction at a certain temperature, the equilibrium constant at a certain temperature and a pressure.

2. *Semantic Relationships among Modeling Objects*

Semantic relationships provide links between the various modeling elements, described above, and give rise to a network known as *semantic* *network*. Then, the description of a process is given by a semantic network. For more information on the nature of the semantic relationships and their role in modeling complex objects, the reader is referred to the more extensive discussion on the subject in the first chapter in the volume. For describing the structural aspects of a process plant, the following semantic relationships from MODEL.LA. are available in HDL. In the descriptions below, semantic relationships are written in boldface and italics. The class name starts with the uppercase letter, e.g., `Reactor`, and the instance name with a definite or indefinite article followed by its class name, e.g., `aReactor` or `theReactor`. A parenthesis represents a set, e.g., (`aReactor, aSeparator`).

is-a. The ***is-a*** relationship indicates the parent/ child relationship of two classes, e.g., `Plant` ***is-a*** `GenericUnit`.

is-a-member-of. The ***is-a-member-of*** relationship indicates the relationship between a class and its instance, e.g., `aReactor` ***is-a-member-of*** `Reactor`.

is-composed-of. The ***is-composed-of*** relationship is used to identify the modeling objects that are parts of another object, e.g., `aHeatExchanger` ***is-composed-of*** (`aTube`, `aBundle`, `aShell`).

is-part-of. The ***is-part-of*** relationship is the inverse relationship of the ***is-composed-of*** relationship. For example, `aShell` ***is-part-of*** `aHeatExchanger`.

is-attached-to. The ***is-attached-to*** relatinship provides the means for connecting process units and streams. The connection takes place through a linkage object called a port, e.g., `aPort` ***is-attached-to*** `aStream`.

is-connected-by. The ***is-connected-by*** relationship is the inverse relationship of the ***is-attached-to*** relationship, e.g., `aGenericUnit` ***is-connected-by*** `aStream`.

In addition to the above, the following semantic relationships were found to be valuable and were introduced into the HDL.

is-a-model-of. The ***is-a-model-of*** relationship indicates the relationship between a user interface and a model. An instance of a `GenericUnit` or its descendant classes represents the model of the process unit. For example, `ReactionSystem` has reaction information and has the operations to compute the heat of reaction and equilibrium constant, etc. A graphic unit associated with the reaction system represents the iconic structure of the reaction system, e.g., the number of input ports and the number of output ports. Figure 25 shows this relationship.

is-alternative-of. The ***is-alternative-of*** relationship relates a flowsheet to another alternative flowsheet. Both objects are of the same class and satisfy the same functions. However, they may have different structures, which result from different design decisions. Alternatives are used to keep track of the evolution of a flowsheet by recording the changes that are made to them.

B. Modeling the Design Tasks

In the previous section, we presented the modeling classes that allow one to describe the structure and behavior of chemical processing systems. Such a representation provides and structures the declarative knowledge about the current state of the design.

As a design is a transformation process of functional specifications into realizable physical objects, we need constructs that correspond to these transformations. A design task is a high-level construct that takes an action of transforming a design state to more detailed state, which satisfies the function of the design task. As a design task corresponds to the design action, the task has its own design goal and a design plan about how to accomplish its goal. Specific characteristics of design tasks are presented in this section.

1. *Basic Task Elements*

There are two kind of design tasks:design managers and design agents. A design manager is concerned primarily with organizing the computation, while the details of carrying out the steps are handled by design agents. Figure 17 shows the computational process for the various design tasks contained in HDL. The position of the task in the hierarchy shown in Figure 17 defines the role and the scope of the task during the design process. Figure 18 shows the inheritance relationships among design tasks.

a. Design-Task-Element 1: `GenericManager`. `GenericManager` can be considered a team leader who exercises control over subordinates, each of whom makes decisions within the context of its own bounded sub tasks. When subordinates encounter insurmountable difficulties in their sub tasks, some overall strategy may be introduced by the team leader to improve interactions between the subordinates.

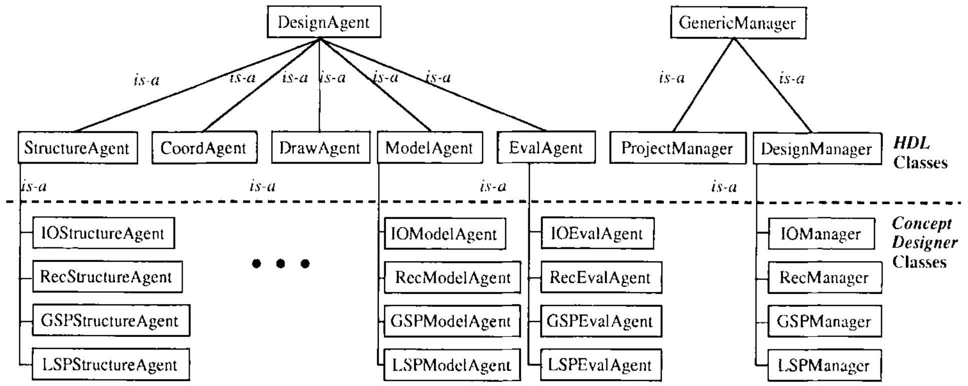


FIG. 18. The class tree of the design managers and the design agents in *ConceptDesigner*. The ancestor classes of `DesignAgent` and `GenericManager` exist, but not shown here. HDL classes are shown above the dotted line.

(i) *Design-Task-Element 1.1:* `DesignManager`. `DesignManager` is an abstract task that can be used to create a specific design manager. The classes that are descendants of the design manager can refine a process flowsheet from less detailed levels to more detailed levels. Each design manager carries out a functional transformation as shown below:

$$\texttt{aDesignManager(aFlowsheet}_i\texttt{)} \Rightarrow \texttt{aFlowsheet}_{i+1}$$

where subscript i denotes the abstraction level. For instance, `RecycleManager` refines a flowsheet at the input/output structure level into a flowsheet at the recycle structure level as follows:

$$\texttt{aRecycleManager(aIOStructure)} \Rightarrow \texttt{aRecycleStructure.}$$

(ii) *Design-Task-Element 1.2:* `ProjectManager`. `ProjectManager` is a task that organizes and coordinates the activities of design managers for a design project. Figure 17 shows `ProjectManager` at the top of the hierarchy among the design tasks. This hierarchy defines the roles of design tasks, while the class hierarchy defines the inheritance relationships.

b. Design-Task-Element 2: `DesignAgent`. `DesignAgent` is an abstract task that has a domain-specific knowledge. Specific design knowledge is represented as a design plan for each design agent. The following are subclasses of the `DesignAgent` class. Design agents 2.1–2.3 are sufficient to synthesize a structure and present it to the user in a very friendly manner. It should be noted that there is a certain order of executing these agents; `StructureAgent` first, `CoordAgent` second, and `DrawAgent`. The structural synthesis is completed by these agents. Agents 2.4 and 2.5 set up the material and energy balance and compute the economic potential of the flowsheet.

(i) *Design-Task-Element 2.1:* `StructureAgent`. `StructureAgent` is an abstract task that decomposes a given system into a set of subsystems and coordinates subsystems so that most of interconnections among sub systems are identified. The specific design knowledge for the decomposition and the coordination should be provided when the subclass is created for specific structure design. For instance, `aIOStructureAgent` shown in Fig. 17 has a knowledge and a design plan on how to synthesize an input/output structure. The `RecStructureAgent` shown in Fig. 17 has a knowledge and a design plan about how to sythesize a recycle structure. The interaction with the user is needed only when the agent does not have heuristic rules about the decision.

(ii) *Design-Task-Element 2.2:* `CoordAgent`. `CoordAgent` is an abstract task that establishes the connections among the `Ports` of the `GenericUnits` that do not have connections. Thus, this agent supplements the `StructureAgent` in terms of coordination.

(iii) *Design-Task-Element 2.3:* `DrawAgent`. `DrawAgent` is an abstract task that transforms a `GenericUnit` into an icon associated with the given `GenericUnit`. It also draws the streams that connect the ports of a generic unit to the other ports of generic unit based on the connection information of each port. It is a precondition that the connection information has been already identified by a `StructureAgent` and a `CoordAgent`.

(iv) *Design-Task-Element 2.4:* `ModelAgent`. `ModelAgent` is an abstract task that is in charge of the analysis. After the structure has been synthesized by the `StructureAgent`, a `ModelAgent` sets up and solves the material and energy balances, based on the synthesized structure. When design specifications are needed to finish the analysis, the agent may ask the user for these specifications.

(v) *Design-Task-Element 2.5:* `EvalAgent`. `EvalAgent` is an abstract task that is in charge of the evaluation. It computes the economic potential of the current process flowsheet.

2. Semantic Relationships among Design Tasks

The following list of semantics establishes the requisite relationships among the various design-task elements described above. The resulting semantic network (nodes are design tasks, edges are semantic relationships) models the overall design methodology.

decompose. The ***decompose*** relationship is used to decompose a boundary system into a set of subsystems. The syntax for this relationship is

`aStructureAgent` ***decompose*** `aBoundarySystem`

For example, `aRecStructureAgent` ***decompose*** `aPlantSystem`.

is-decomposed-into. The ***is-decomposed-into*** relationship is used to indicate the results of applying ***decompose*** relationship. The syntax is

`aGenericUnit` ***is-decomposed-into*** (a list of `aGenericUnits`)

For example, `aPlantSystem` ***is-decomposed-into*** (`aReactionSystem`, `aSeparationSystem`).

transform. The ***transform*** relationship is used to transform a list of generic units into a list of graphic units. Each graphic unit becomes associated with the corresponding generic unit. The syntax is

`aDrawAgent` ***transform*** (a list of `aGenericUnits`)

For example, `aIODrawAgent` ***transform*** (`aPlantSystem`).

is-transformed-into. The ***is-transformed-into*** relationship is used to indicate the results of applying transform relationship. The syntax is

(a list of `aGenericUnits`) ***is-transformed-into***
(a list of `aGraphicUnits`)

For example, (`aReactionSystem`) ***is-transformed-into*** (`aGraphicUnit`).

compute. The ***compute*** relationship is used to compute the material and energy balances for a flowsheet. A `ModelAgent` computes the material and energy balances for all the process units that exist on the flowsheet. The syntax is

`aModelAgent` ***compute*** `aFlowsheet`

For example, `aRecModelAgent` ***compute*** `aRecycleFlowsheet`.

evaluate. The ***evaluate*** relationship is used to evaluate the economic potential of a flowsheet. An `EvalAgent` evaluates an economic potential of a `Flowsheet` by sending the message "computeCost" to all the process units which exist on the flowsheet. The syntax is

`aEvalAgent` ***evaluate*** `aFlowsheet`

For example, `aRecEvalAgent` ***evaluate*** `aRecycleFlowsheet`.

refine. The ***refine*** relationship is used by a design manager to refine an instance of `aGenericSystem` at the current abstraction level into a more detailed flowsheet at the next abstraction level. The syntax is

`aDesignManager` ***refine*** `aGenericSystem`

For example, `aRecManager` ***refine*** `aPlantSystem` at the input/ output structure.

is-refined-into. The ***is-refined-into*** relationship is used to indicate the results of applying ***refine*** relationship. The syntax is

`aGenericSystem` ***is-refined-into*** `aFlowsheet`

For example, `aPlantSystem` at the input/ output structure level ***is-refined-into*** `aFlowsheet` at the recycle structure level.

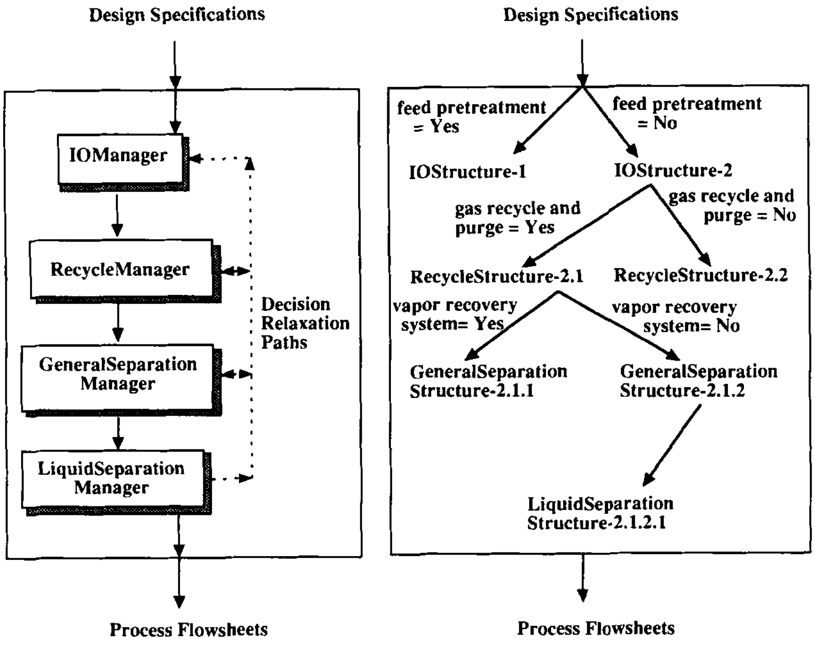

Design Specifications
IOManager
RecycleManager
GeneralSeparation Manager
LiquidSeparation Manager
Decision Relaxation Paths
Process Flowsheets
Design Specifications
feed pretreatment = Yes
feed pretreatment = No
IOStructure-1
IOStructure-2
gas recycle and purge = Yes
gas recycle and purge = No
RecycleStructure-2.1
RecycleStructure-2.2
vapor recovery system= Yes
vapor recovery system= No
GeneralSeparation Structure-2.1.1
GeneralSeparation Structure-2.1.2
LiquidSeparation Structure-2.1.2.1
Process Flowsheets

FIG. 19. Generation of design alternatives by relaxing design decisions.

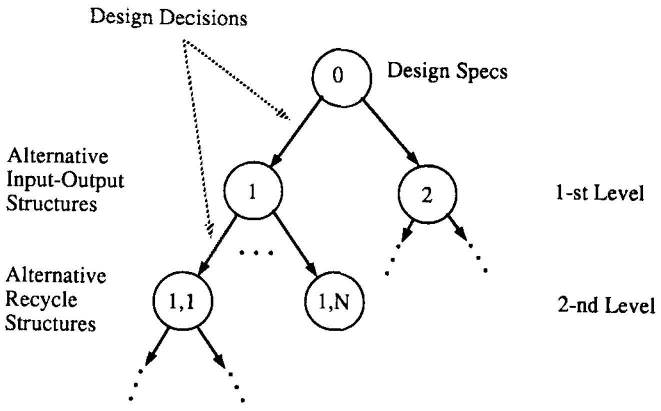

Design Decisions
0
Design Specs
Alternative Input-Output Structures
1
2
1-st Level
Alternative Recycle Structures
1,1
1,N
2-nd Level

FIG. 20. The tree data structure maintaining design alternatives.

C. ELEMENTS FOR HUMAN–MACHINE INTERACTION

In order to communicate with the user, a computational procedure should possess very informative user interfaces, such as those described in following paragraphs, which are part of the *ConceptDesigner* (see Section IV; see also Figs. 22–25).

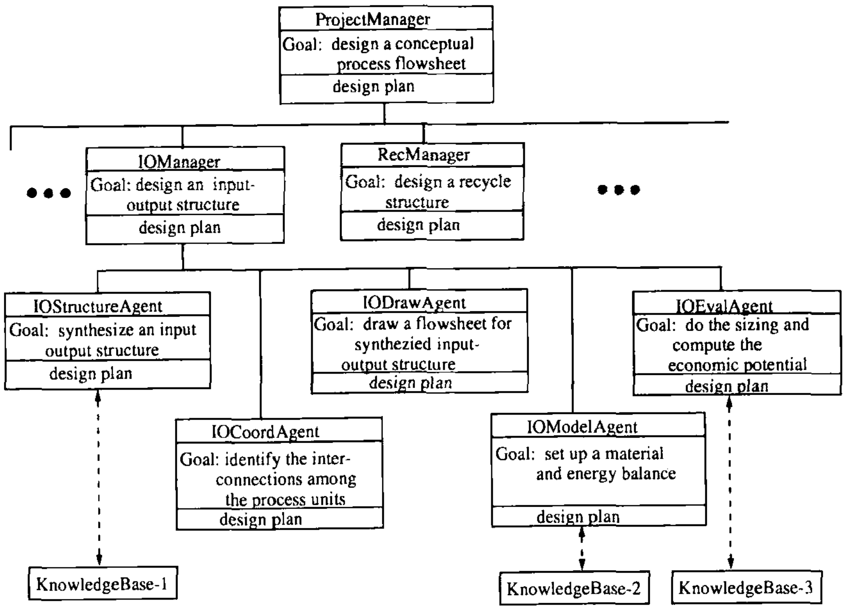

ProjectManager
Goal: design a conceptual process flowsheet
design plan
IOManager
Goal: design an input-output structure
design plan
RecManager
Goal: design a recycle structure
design plan
IOStructureAgent
Goal: synthesize an input output structure
design plan
IODrawAgent
Goal: draw a flowsheet for synthezied input-output structure
design plan
IOEvalAgent
Goal: do the sizing and compute the economic potential
design plan
IOCoordAgent
Goal: identify the inter-connections among the process units
design plan
IOModelAgent
Goal: set up a material and energy balance
design plan
KnowledgeBase-1
KnowledgeBase-2
KnowledgeBase-3

FIG. 21. Design task hierarchy in *ConceptDesigner*.

1. *Human Interaction Element 1:* `MainWindow`

This is an abstraction for the main interface. It provides several capabilities to define a design project. A user can define new project, load an old project, or save a current project into a file through `MainWindow`. The user can also define a new physical property database, load an existing database, modify the loaded database, and save the loaded database to a file. The user can specify the design mode: an automatic design mode, or an interactive design mode. In the automatic design mode, the design will be done by the system and the system will ask questions when they are needed. In the interactive design mode, the user can control the design process by executing each design task.

2. *Human Interaction Element 2:* `LayoutWindow`

This is an abstraction for the work space. A process flowsheet is developed and presented at the `LayoutWindow`. All design tasks are applied to an instance of `LayoutWindow` to transform a current process flowsheet. A snapshot of the transition in the process flowsheet is a design state at the moment. Figures 22 and 23 show the snapshots of a `LayoutWindow` during the design of hydrodealkylation-of-toluene process.

The elements described above are places where new modeling elements are designed and their values are displayed. The displayed elements are associated with the modeling elements and provide the user interface through which the user can access all the information generated during the design process. Those elements are presented below.

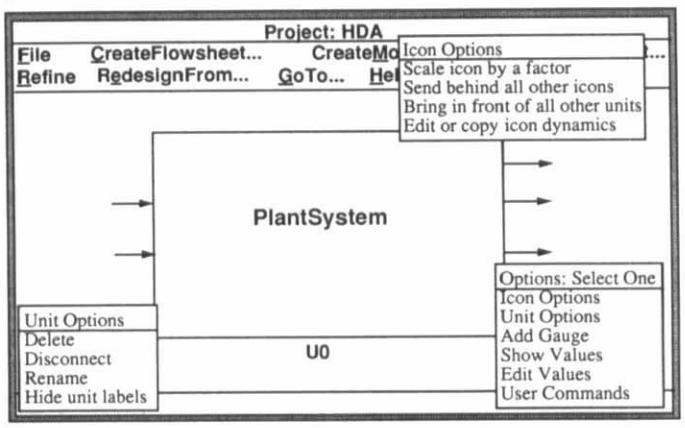
3. *Human Interaction Element 3:* `GraphicUnit`

This is an abstraction associated with a `GenericUnit.GraphicUnit`, displayed on the display area, is a means through which the user can communicate with the `GenericUnit`. A user can see the values of the design parameters. The user can also modify the graphic elements of the process unit, i.e., move the icon to the different place, rotate, or zoom. In Fig. 23b, `aGraphicUnit` displays the information of the associated distillation unit.

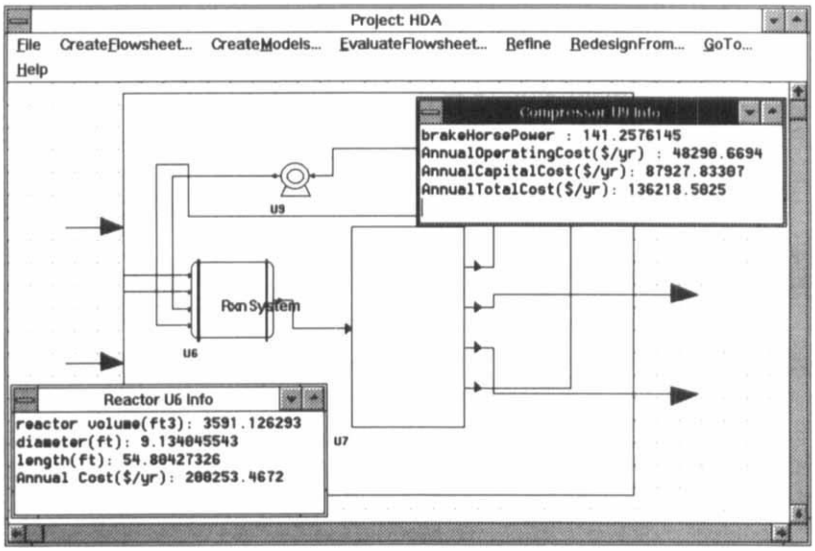
4. *Human Interaction Element 4:* `GraphicPort`

This is an abstraction associated with a `Port`. A user can modify the `GraphicPort` graphically, i.e., move, rotate, or zoom. The user can also see the values of `aPort` associated with the `GraphicPort`. In Fig. 23a, a `GraphicPort` shows the information of the associated port.

a



b

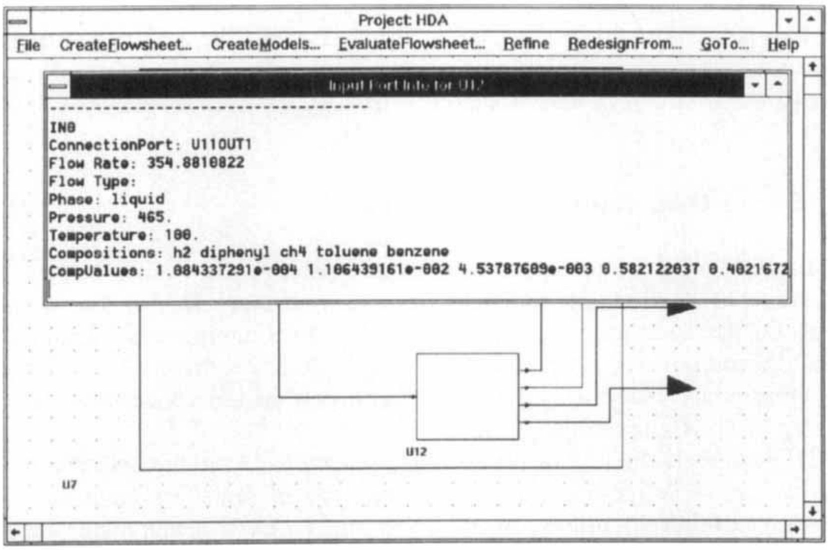


```
brakeHorsePower : 141.2576145
AnnualOperatingCost($/yr) : 48290.6694
AnnualCapitalCost($/yr): 87927.83307
AnnualTotalCost($/yr): 136218.5025
```

```
reactor volume(ft3): 3591.126293
diameter(ft): 9.134045543
length(ft): 54.80427326
Annual Cost($/yr): 200253.4672
```

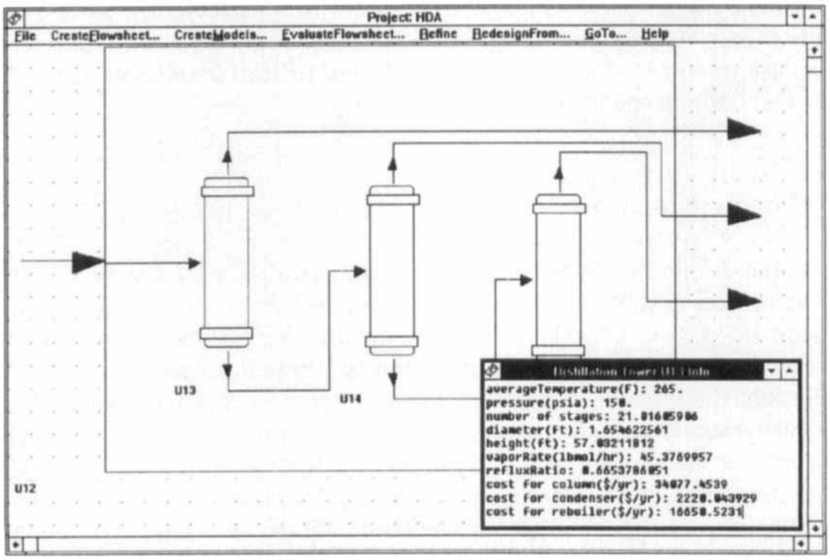
FIG. 22. User interfaces at (a) input/output structure level and (b) recycle structure level.

a


Project: HDA
File CreateFlowsheet... CreateModels... EvaluateFlowsheet... Refine RedesignFrom... GoTo... Help
Input Port Info for U12
U12
U7

```
-----------------------------------
IN0
ConnectionPort: U11OUT1
Flow Rate: 354.8810822
Flow Type:
Phase: liquid
Pressure: 465.
Temperature: 100.
Compositions: h2 diphenyl ch4 toluene benzene
CompValues: 1.084337291e-004 1.106439161e-002 4.53787609e-003 0.582122037 0.4021672
```

b


Project: HDA
File CreateFlowsheet... CreateModels... EvaluateFlowsheet... Refine RedesignFrom... GoTo... Help
U13
U14
U12
Distillation Tower U13 Info

```
averageTemperature(F): 265.
pressure(psia): 150.
number of stages: 21.01605906
diameter(ft): 1.654622561
height(ft): 57.03211012
vaporRate(lbmol/hr): 45.3769957
refluxRatio: 0.6653706051
cost for column($/yr): 34077.4539
cost for condenser($/yr): 2220.043929
cost for reboiler($/yr): 16650.5231
```

FIG. 23. User interfaces at (a) general separation structure level and (b) liquid separation structure level.

5. *Human Interaction Element 5:* `GraphicStream`

This is the abstraction associated with `Stream`. Like a `GraphicUnit`, the user can modify a `GraphicStream` graphically. The user can also see the values of a `aStream` associated with a `GraphicStream`.

D. Object-Oriented Failure Handling

In producing a design, the software system will design a process flowsheet in a direction specified by its design plan. During the design process, the user may encounter obstacles or commit errors. Examples include violations of design constraints and numerical errors. These errors or failures are likely to occur more frequently in the conceptual design where HDL is intended to be used.

HDL models the way of producing a design as hierarchically structured design tasks. A hierarchically structured model leads to hierarchically structured failure handling. As shown in Fig. 17, each design agent has an explicitly defined role. The explicit structure of the model of the design process makes it possible to know what to do in the even of failure. When a failure occurs, the relevant information becomes immediately accessible due to the explicit localization of the failure. In some case, the system can guide the user to the point where the control should backtrack. Therefore, it is easy to correct the local error, and then to start a redesign from the agent which is responsible for the error.

E. Management of Design Alternatives

As the design decisions committed at the conceptual design fix 70–80% of the project cost, it is very important to design the economically optimal process flowsheet. A mechanism for managing alternatives has been developed, based on the characteristics of the conceptual process design itself: evolutionary, iterative decisionmaking process. The mechanism helps the user generate design alternatives by relaxing the previous design decisions.

The user can use the mechanism in the following way. First, he or she develops the base-case design which does not violate any process constraints. Then, the user improves the base-case design in the evolutionary manner by relaxing the decisions that look relatively promising in terms of contribution to the economic potential of the process flowsheet. Suppose we are working on the recycle structure level and we find one design decision about the gas recycle difficult to make because there are no heuristics available for the decision. We must then make a decision to recycle and purge gas components. After we complete the design of the recycle structure by deciding the values for the design variables and computing the economic potential, we can go back to the input/output structure level and restart the design. As all design tasks (including design managers and design agents) work as independent objects, it is a simple matter to go back to any abstraction level and start a redesign. The loaded design managers do the design in the same way except the decision about the gas recycle (e.g., we will choose not to recycle gas components). Figure 19 shows the design alternative generation process by relaxing design decisions. Figure 20 shows the tree structure which is dynamically generated by *ConceptDesigner* and is used to maintain design alternatives. The design decisions are recorded to a file and retrieved from the file along with design alternative. This helps a user identify the design decisions associated with a design alternative and relax them.

IV. ConceptDesigner: The Software Implementation

ConceptDesigner (Han, 1994) is the software system that implements the computational model, described in Section II, of the design process, using the linguistic constructs of HDL. The *ConceptDesigner* was designed to automate large segments of the design process with minimal interaction with the user. Nevertheless, a highly informative user-interface allows the human designer to monitor continuously the progress of the evolving design, and to override the automatic mode with a manual interactive mode of decisionmaking.

A. Overall Architecture

ConceptDesigner is an object-oriented software system, which was built entirely on the top of the HDL design language. Figure 18 shows the complete set of HDL objects that *ConceptDesigner* used to create its own customized design tasks. In the design of *ConceptDesigner*, there are one-to-one mappings between the design-oriented tasks, identified in Section II, and the modeling elements of HDL, discussed in Section III. In other words, the structure of the software objects in *ConceptDesigner* is in direct and explicit correspondence to the design methodology we have

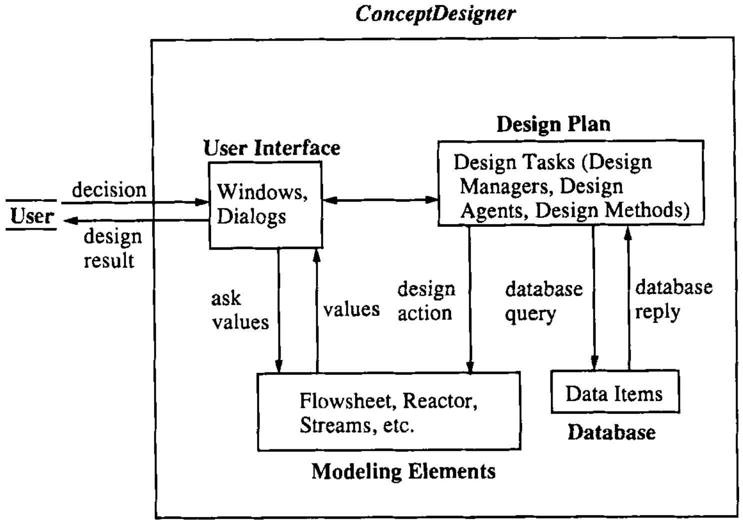


FIG. 24. Functional modules of *ConceptDesigner*.

adopted for the synthesis of process flowsheets. For example, Fig. 21 shows the hierarchical interrelationship among various objects (provided by HDL) in *ConceptDesigner*. Notice the one-to-one correspondence of the objects in Fig. 21, to the design tasks of the goal structure in Fig. 5, which models the design methodology. For example, the object `Project-Manager` in Fig. 21 corresponds to the goal"design a conceptual chemical process" of Fig. 5. Similarly, we can see the following correspondence; objects, `IOManager, RecManager` in Fig. 21 correspond to goals "design an input/ output structure" and "design a recycle structure" of Fig. 5. Furthermore, the objects in Fig. 21, `IOStructureAgent, IOModelA-gent`, and `IOEvalAgent`, correspond to the goals of Fig. 9, "synthesize an input/ output structure," "analyze an input/ output structure," and "evaluate an input/ output structure."

Figure 24 shows the four major modules of *ConceptDesigner*. Figure 25 illustrates the semantic relationships among major objects in those modules.

1. Design Plan Module

This contains all the design-task modeling elements of HDL, and has structured them, through message passing methods, in such a way that

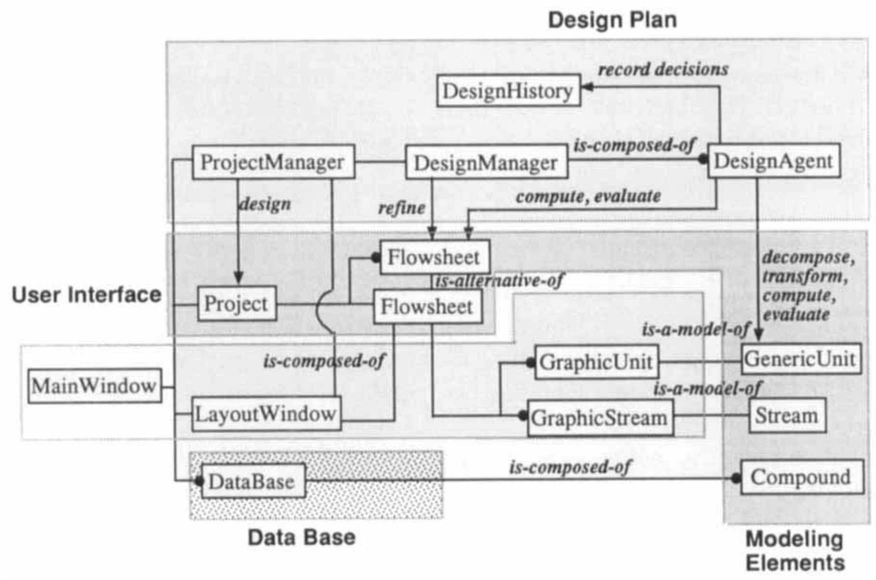


FIG. 25. Object model for *ConceptDesigner*.

they implement the goal structures of Figs. 8, 9, 10, 12, and 13, which model the design methodology. Thus, the `ProjectManager` can access information about the process design at any stage of its development, possess a "design plan" and calls on the `DesignManagers` to execute it. These managers are the softward objects: `IOManager, RecManager, GSPManager`, and `LSPManager` (see Fig. 17). Each of these managers possesses the data and methods to carry out the design of input/ output structure, recycle structure, generalized separation structure, and liquid separation subsystem, respectively. They carry out their corresponding design tasks through the activation of five software objects, the `DesignA-gents`, with common structure but different implementations. Figure 17 shows the structure of the specific `DesignAgents`, used at various stages of the design processed. The various `DesignAgents` contain specific methods, which are used to define the structure of processing systems, set up mathematical models, solve mathematical models, do sizing and costing of units, call on databases for physical properties, make decisions, etc. These methods are implemented through algorithms that contain symbolic manipulations, numerical procedures, or/ and production rule-based systems.

2. *Modeling Objects Module*

This contains all the modeling elements that HDL uses to provide contextual, hierarchical and multi-view representation of process flowsheets (see Section III, A).

3. *User Interface Module*

This contains a broad variety of classes that the *ConceptDesigner* acquired from *x-Kit* (a computer-aided environment for the expeditious tailoring of graphic-oriented software modules, developed at MIT-LISPE). These classes provide the following capabilities: construct icons for abstract or detailed processing units; construct iconic flowsheets; provide animation and dynamic views to the process flowsheets; dialog windows, tables, lists, control buttons, and other user interface elements; and graphs, and charts for the presentation of data. *x-Kit* has also provided the following facilities that can be used for linking *ConceptDesigner* with external programs and databases:

(a) *File-Linker*. A high-level editor to configure the mapping of data between the objects of *ConceptDesigner* and external ASCII (American Standard Code for Information Interchange) files. These mappings can be used to transfer data to input files for an external FORTRAN program, or retrieve data from output files that carry the results of the FORTRAN program.
(b) *C-function Linker*. A program that uses dynamic link libraries and allows any object-method to be implemented through an external C function.
(c) *Database-Linker*. A high-level editor to configure the mapping of data between the tables of external relational databases and the *ConceptDesigner*'s objects. Among the many graphic user interfaces, the *LayoutWindow* occupies a central position, since this is the main canvas where the flowsheet is shown. Various instances of the *LayoutWindow* depict the input/ output, recycle, generalized separation, liquid separation, and other abstractions of the evolving flowsheet (see Figs. 22 and 23).

Figure 26 provides a different view of the *ConceptDesigner*'s structure. It shows that every project generates a series of distinct process flowsheets, all of which are handled by a file management system, which is organized in a hierarchical tree to reflect the evolution of the design and the various alternative designs. It also depicts the various sources of data

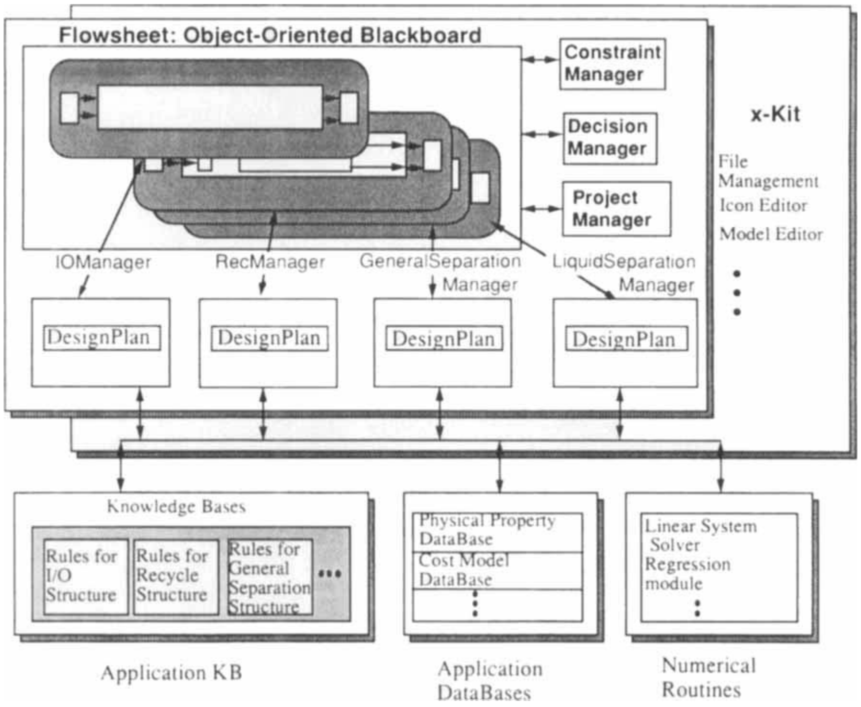


FIG. 26. Overall architecture of *ConceptDesigner*.

used and the types of knowledge bases and numerical procedures employed.

B. Implementation Details

HDL has been implemented in Actor, which is an object-oriented language. The user interfaces have been developed from Microsoft Windows 3.1. *ConceptDesigner* can be run in any PC (personal computer)-compatible computer.

Conceptual process design is a complex process. As the design goes through several abstraction levels and generates many design alternatives at each abstraction level, a huge space is needed to store all information which is not usually available in PC or workstation environment. *ConceptDesigner* uses a dynamic memory allocation and a paging to resolve these memory problems. When an object is temporarily created, a certain amount of memory is allocated to the object. When the object is no longer needed, the allocated memory is immediately recovered by a garbage collection system. Another technique is a paging. After a process flowsheet is designed and the design process moves to the next abstraction level, the current process flowsheet is automatically saved to a file with a unique name. The memory consumed by the flowsheet is automatically recovered. These techniques enable *ConceptDesigner* to deal with large design cases.

V. Summary

In this chapter, we have tried to present a detailed approach for automating a complex design methodology. Three requirements are considered pivotal for the execution of projects, namely:

1. An explicit and formal model of the design methodology must be created. This model is the essence of the computational procedure to be developed.
2. A modeling language is needed to provide multi-faceted representation of the evolving design artifact, and represent the design tasks and their semantic relationships.
3. An object-oriented implementation of the software system is absolutely necessary.

The *ConceptDesigner* is a software system that was developed to automate large segments of Douglas' methodology for the synthesis of conceptual process designs. It satisfies all three of the above requirements. The most important lesson learned can be summarized as follows:

1. Hierarchical planning is an essential ingredient for modeling explicitly a design methodology. It sketches the general specs of the intermediate design milestones, without restricting the design activities too severely. Goal structures based on the concept of abstract refinement offer a very good model for describing the design plan.

2. Goal structures based on the notion of a unified, global transformation represent a very attractive generic model for the representation of the design methodology that transforms the artifact from an intermediate design milestone to the next. Such goal structures can accommodate equally well depth-first evolution of designs (such as the one suggested in Douglas' methodology), or exhaustive searches through implicit enumeration of design alternatives (such as those used in the MILP or MINLP formulation of process synthesis problems).

3. We cannot overstate the significance of modeling languages. A well-defined language captures explicitly, through its modeling elements,

domain-specific knowledge, and structures in a generic manner most of the design tasks.

4. The software that automates segments of a design activity should represent an explicit map of the design methodology. This is a necessary and sufficient condition in order to produce a software system which is (a) maintainable and (b) open to modifications and extensions with new knowledge.

References

Balzer, R., Goldman, N., and Wile, D., On the transformational implementation approach to programming. *Proc. Int. Conf. Software Eng., 2nd*, p. 337 (976).

Chandrasekaran, B., Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* **1**, 3 (1986).

Douglas, J. M., As hierarchical decision procedure for process synthesis. *AIChE J.* **31**(3), 353 (1985).

Douglas, J. M., "Conceptual Design of Chemical Processes." McGraw-Hill, New York, 1988.

Douglas, J. M., Synthesis of multistep reaction processes. *In* "Foundations of Computer-Aided Process Design" (J. J. Siirola, I. E. Grossmann, and G. Stephanopoulos, eds.), p. 79. CACHE Corp., Austin, TX, and Elsevier, New York, 1989.

Edgar, T. F., and Himmelblau, D. M., "Optimization of Chemical Processes," pp. 413–422. McGraw-Hill, New York, 1988.

Grossmann, I. E., Mixed-integer programming approach for the synthesis of integrated process flowsheets. *Comput. Chem. Eng.* **9**, 463 (1985).

Grossmann, I. E., MINLP optimization strategies and algorithms for process synthesis. *In* "Foundations of Computer-Aided Process Design" (J. J. Siirola, I. E. Grossmann, and G. Stephanopoulos, eds.), p. 105. CACHE Corp., Austin, TX, and Elsevier, New York, 1989.

Han, C., Human-aided, computer-based design paradigm: The automation of conceptual process design. Ph.D. Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA (1994).

Johnston, J., Synthesis of control structures for complete chemical plants. Ph.D. Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA (1991).

Kirkwood, R. L., PIP-process invention procedure, a prototype expert system for synthesizing chemical process flowsheets. Ph.D. Thesis, Department of Chemical Engineering, University of Massachusetts, Amherst (1987).

Kirkwood, R. L., Locke, M. H., and Douglas, J. M., A prototype expert system for synthesizing chemical process flowsheets. *Comput. Chem. Eng.* **12**, 329 (1988).

Kocis, G. R., and Grossmann, I. E., A modelling/decomposition strategy for MINLP optimization of process flowsheets. *Comput. Chem. Eng.* **13**, 797 (1989).

Kritikos, T., A model for process design automation. Ph.D. Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA (1991).

Mahalec, V., and Motard, R. L., Procedures for the initial design of chemical processing systems. *Comput. Chem. Eng.* **1**, 57 (1977).

Malone, M. F., Glinos, K., Marquez, F. E., and Douglas, J., Simple, analytical criteria for the sequencing of distillation columns. *AIChE J.* **31**, 683 (1985).

McKenna, T. F., and Malone, M. F., Polymer process design. I. Continuous production of chain-growth homopolymers. *Comput. Chem. Eng.* **14**, 1127 (1990).

Mitchell, T., Steinberg, L., Reid, G., Schooley, P., Jacobs, H., and Kelly, V., Representations for reasoning about digital circuits. *Proc. IJCAI*-81 (1981).

Mostow, J., Toward better models of the design process. *AI Mag.* Spring, p. 44 (1985).

Newell, A., The knowledge level. *AI J.* **19**(2), 87 (1982).

Piela, P. C., ASCEND—An object-oriented environment for the development of quantitative models. Ph.D. Thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA (1989).

Powers, G. J., Heuristic synthesis in process development. *Chem. Eng. Prog.* **68**, 88 (1972).

Quantrille, T. E., Liu, Y. A., "Artificial Intelligence in Chemical Engineering." Academic Press, San Diego, CA, 1991.

Rajagopal, S., Ng, K. M., and Douglas, J. M., A hierarchical procedure for the conceptual design of solids processes. *Comput. Chem. Eng.* **16**, 675 (1992).

Rich, E., Knight, K., "Artificial Intelligence," 2nd ed. McGraw-Hill, New York, 1991.

Rossiter, A. P., and Douglas, J. M., Design and optimization of solids processes. Part 1. A hierarchical decision procedure for process synthesis of solids systems. *Chem. Eng. Res. Des.* **64**, 175 (1986a).

Rossiter, A. P., and Douglas, J. M., Design and optimization of solids processes. Part 2. Optimization of crystallizer, centrifuge and dryer systems. *Chem. Eng. Res. Des.* **64**, 184 (1986b).

Rudd, D. F., Powers, G. J., and Siirola, J. J., "Process Synthesis." Prentice-Hall, Englewood Cliffs, NJ, 1973.

Scherlis, W., and Scott, D., First steps towards inferential programming. *IFIP Congr. '83* (1983).

Siirola, J. J., Powers, G. J., and Rudd, D. F., Synthesis of system design. II. Toward a process concept generator. *AIChE J.* **17**, 677 (1971).

Siletti, C. A., Computer-aided design of protein recovery processes. Ph.D. Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA (1988).

Siletti, C. A., and Stephanopoulos, G., BioSep designer: A knowledge-based process synthesizer for bioseparations. In "Artificial Intelligence Approaches in Engineering Design" (C. Tong and D. Sriram, eds.). Vol. I, p. 295. Academic Press, San Diego, CA, 1992.

Stefik, M., Bobrow, D., Bell, A., Brown, H., Conway, L., and Tong, C., The partitioning of concepts in digital system design. *Proc. Conf. Adv. Res. VLSI*, p. 43 (1982).

Stephanopoulos, G., Artificial intelligence and symbolic computing. In "Foundations of Computer-Aided Process Design" J. J. Siirola, I. E. Grossmann, and G. Stephanopoulos, eds.), p. 21. CACHE Corp., Austin, TX, and Elsevier, New York, 1989.

Stephanopoulos, G., Henning, G., and Leone, H., MODEL.LA.:A modeling language for process engineering: Part I. The formal framework. *Comput. Chem. Eng.* **14**, 813 (1990a).

Stephanopoulos, G., Henning, G., and Leone, H., MODEL.LA.:A modeling language for process engineering: Part II. Multifaceted modeling of processing systems. *Comput. Chem. Eng.* **14**, 847 (1990b).

Stephanopoulos, G., Han, C., Linninger, A., Ali, S., and Stephanopoulos, E., Concept of ZAP (Zero Avoidable Pollution) in the synthesis and evaluation of batch pharmaceutical processes. Paper presented at *Ann. AIChE Meet.*, San Francisco (1994).

Tong, C., Toward an engineering science of knowledge-based design. *AI Eng.* **2**, 133 (1987).

Tong, C., and Sriram, D., "Artificial Intelligence in Engineering Design," Vol. I, p. 9. Academic Press, San Diego, CA, 1992.