Generation of Complex Reaction Systems through a Specialized Computer Language

Suzanne **E. Prickett and Michael L. Mavrovouniotis***
*Chemical Engineering Department, Northwestern University, Evanston, IL 60208-3120

Abstract

We describe an approach for the computational modeling of complex reaction systems, based on a language that allows the description of general types of chemical reactions. These descriptions are compiled and automatically executed to generate the entire reaction network. The language provides a general method for describing reactions–not limited to a specific class of chemical transformations. Reactions are described using a sequence of commands to characterize the reaction site, the transformation of the reactants to products, and, if desired, thermodynamic constraints to determine the dominant steps within each reaction type. These commands allow flexibility in the types of reaction systems that can be analyzed and may also be used to adjust the level of detail within a specific reaction network. The commands were developed to mimic the natural way in which a generic reaction is often described informally.

INTRODUCTION

Complex reaction systems occur in many chemical processes, including fermentation, combustion, polymerization, and petroleum refining. Modeling such systems is difficult because there are large numbers of reactions and reaction intermediates involved (Mavrovouniotis *et al.*, 1993). Better models of complex reaction systems can lead to improvements in the design, operation, and control of these chemical processes. Improved accuracy of chemical analysis techniques and increased computational capabilities are gradually enabling computational study of reaction networks in their full detail.

The computational approach described here is based on a language that allows the description of general types of chemical reactions, which are then used to generate the entire reaction network for any given feed. The distinguishing feature of this work is that it can be used for *any* type of reaction system. Reactions are described using a sequence of commands to characterize the reaction site, the transformation of the reactants to products, and, if desired, constraints to determine the dominant steps within each reaction type. These commands allow flexibility in the types of reaction systems which may be analyzed and in adjusting the level of detail of the network.

THE REACTION DESCRIPTION LANGUAGE

The Reaction Description Language, RDL, is a computer language developed to describe generic types of chemical reactions. Using the keywords of RDL, commands may be formed that locate the reaction site, manipulate the reactant to form the product, and determine whether a reaction may be applied to a specific reaction site. The syntax and the types of operators in RDL mimic the way in which a generic reaction is often described informally, Thus, the reaction descriptions are usually self-documenting. Modification or addition of reaction types is accomplished easily and transparently.

The permitted keywords of the language, shown in Fig. 1, include operators and item descriptors. An operator receives a set of arguments and performs a particular task. For example, the operator *disconnect* ensures that its two arguments, which must represent portions of a molecule, are adjacent to each other and disconnects them. An item descriptor refers to a portion of the compound or network (e.g., reaction, positive-atom, or reactant). Item descriptors are formed using prefixes, items, and suffixes (Fig. 1). Valid item descriptors include "negative-atom", "aromatic-rings", and "triple-bond". The combination of certain prefixes and items is undefined; the term "positive-bond" has no meaning and is assumed to represent a variable. A variable is a collection of characters that is defined by the user with an assignment operator, *Label-site* or *Set-var.* A user-defined variable cannot coincide with an operator or legal item descriptor; in effect, the keywords of RDL are *reserved.* The production rules of RDL, in BNF notation, are listed in Fig. 2. A reaction description is formed by a set of statements, or commands, enclosed by braces.

Keyword Category		Definition
assignment-op	→	**label-site | set-var**
pruning-op	→	**require | forbid**
test-op	→	**if**
search-net-op	→	**search-network-for**
set-op	→	**set**
find-op	→	**find | find-all | find-unique**
find-test-op	→	**find-exactly | find-less-than | find-more-than**
transform-op1	→	**add-charge | subtract-charge | increase-order-of | decrease-order-of**
transform-op2	→	**connect | disconnect**
predicate-op	→	**aromatic | hydrocarbon | neutral | primary**
rel-op	→	**equal | greater-than | less-than | more-than | at-least | at-most**
arithmetic-op	→	**times | plus | minus | divided-by**
prep-op1	→	**attached-to | belonging-to**
prep-op2	→	**upto-sphere | in-sphere**
prep-op3	→	**connecting**
rxn-op	→	**number-of-reactions | generation**
item-op	→	**symmetry-number-of | size-of | molecular-weight-of | charge-of | bond-order-of | generation-of**
item-descriptor	→	prefix? || item || suffix?
item	→	**reactant | product | atom | carbon | hydrogen | oxygen | nitrogen | sulfur | phosphorus | bond | ring | chain | reaction**
prefix	→	**positive- | negative- | neutral- | all- | single- | double- | triple- | aromatic- | multiple- | hetero-**
suffix	→	**s | es**

= zero or one   † = zero or more
II= concatenate   I=or

Fig. 1. The keywords of RDL, including operators and item descriptors.

```
reaction                —› { stmt* }
stmt                    —› assignment-stmt | pattern-stmt | pruning-stmt | set-stmt |
                           transform-stmt
expr                    —› arithmetic-expr | find-expr | item-expr | number-expr |
                           partial-prep-expr | preposition-expr | prep-expr | predicate-expr |
                           relative-expr |  slot-expr | transform-expr1 | transform-expr2

Search and Label Phrases
  assignment-stmt       —› ( assignment-op variable item-expr  )
  pattern-stmt          —› ( search-net-op assignment-stmt* pruning-stmt†     )
  find-expr             —› find-item-expr | find-test-expr

Manipulation  Phrases
  transform-stmt        —› (  transform-expr1  I  transform-expr2  )
  transform-expr1       —› trans form-op 1  item-expr
  transform-expr2       —› transform-op2 item-expr  item-expr
  set-stmt              —› ( set-op slot-expr  number-expr  )

Pruning Phrases
  pruning-stint         —› (  pruning-op expr  )
  test-stint            —› ( test-op  expr  expr expr? )
  find-expr             —› find-item-expr  I  lind-test-expr

Miscellaneous Expressions
  arithmetic-expr       —›  (  arithmetic-op number-expr*  )
  find-item-expr        —› ( find-op item-descriptor  partid-prep.cqxt  )
  find-test-expr        —› ( find-test-op  number-expr  item-descriptor partial-prep-expr† )
  item-expr             —› variable I item-descriptor I I find-item-expr
  number-expr           —› test-slmt | find-expr I number-expr I  slot-expr I number
  plmal.prep-expr       —›  prep-expr  item-expr
  predicate-expr        —›  (  predicate-op item-expr  )
  preposition-expr      —›  item-expr  partial-prep-expr
  prep-expr             —› prep-expr1 I prep-expr2  I  prep-expr3
  prep-exprl            —›  prep-op  1
  prep-expr2            —›  prep-IIp2  number-expr
  prep-expr3            —›  prep-op3  item-expr
  relative-expr         —› ( rel-op  number.expr  number.expr  )
  slot-expr             —›  (  item-op item-expr  )
```

 = one or more   ? = zero or one
† = zero or more   I=or

Fig. 2. The production rules of RDL in Backus Naur Form. This is a formal description of the syntax of the language.

As an example, Fig. 3 is a specification for the 1,2-hydride shift reaction. Lines 1–3 of Fig. 3 locate the reaction site and create the variables Carbon 1, Carbon2, and H 1. In locating a reaction site, only topologically unique atoms are found. For the secondary heptyl ion:

Cl H_3–C^+H–C2 H2-CH2-CH2-CH2-CH3 (7)

only two reaction sites are located: (C^+–C l–H) and (C^+–C2–H). The total number of reactions is accounted for in the last line of Fig. 3, (Set Number-of-reactions (Symmetry-number-of H1)). The symmetry number of an atom is the total number of atoms topologically equivalent to it. Thus, the total number of reactions for the reaction site (C^+–C1–H), in compound (1) is three because there are three equivalent hydrogens attached to Cl, Similarly, the total number of reactions for the site (C^+–C2–H) is two because there are two equivalent hydrogens attached to C2.

Lines 4-7 of Fig. 3 are the manipulation statements, which form the product of the reaction. In the secondary heptyl ion example, the compound would be manipulated twice, corresponding to the two reaction sites found, to form a primary heptyl ion and a heptyl ion with the charge on C2, shown below.

C 1^+ H_2–CH_2– C2H2–CH2–CH2–CH2-CH3 (2)

Cl H_3–CH_2– $C2^+$ H-CH2-CH2–CH2-CH3 (3)

The primary ion (2) is discarded when the pruning command on line 8 is applied: (Forbid (Primary Carbon2)), where the variable Carbon2 refers to C 1 in structure (2). This line is included based on the assumption that the formation of primary ions is not favored thermodynamically. Because this command is located after the manipulation statements, Carbc 12 refers to the carbon atom in the product. Thus, the test is applied *after* the product is formed. A more efficient method of rejecting these reaction sites would be to place line 8 before all the manipulation statements. Compile-time optimization can be used to automatically rearrange the execution order of the commands.

Note that the value of a variable shifts from reactants to products at run time. In the discussion above, the variable Carbon2 in lines 1-3 of Fig. 3 refers to a carbon atom in the reactant However, in lines 4-8, Carbon2 refers to a part of the product. In general, a variable refers to the product if it is in a statement that occurs after any manipulation command.

```
1  (Label-site Carbon1 (Find  positive-carbon))
2  (Label-site Carbon2 (Find neutral-carbon attached-to Carbon l))
3  (Label-site H 1 (Find neutral-hydrogen attached-to Carbon2))
4  (Disconnect Carbon2 H1)
5  (Connect  Carbon1 H1)
6  (Subtract-charge Carbon 1)
7  (Set (Charge-of Carbon2) 1)
8  (Forbid (Primary Carbon2))
9  (Set Number-of-reactions (Symmetry-number-of H 1))
   }
```

Fig. 3. RDL commands for the 1,2-hydride shift reaction.

As a second example, a description of β-scission of a carbenium ion (Fig. 4) illustrates the default behavior of phrases within a *Label-site* command. When both variables of the operator *attached-to are* atoms and the phrase occurs within a *Label-site* command, only atoms that are connected by single bonds are located. Thus, the reaction sites found by applying lines 1 through 3 to a compound will only contain carbons connected by single bonds. To specify a reaction site in which the order of the bonds are irrelevant, additional commands must be included. A reaction site composed of a neutral carbon atom connected to a neutral oxygen by a bond of any order could be described by the commands:

(Label-site Cl (Find neutral-carbon))
(Label-site B 1 (Find bond attached-to Cl))
(Label-site Oxyl (Find neutral-oxygen attached-to B 1))

The default behavior within the Label-site commands was developed based on the assumption that multiple bonds play a major role in the types of reactions a compound may undergo. Thus, in describing a reaction site, one generally states the presence of these items explicitly.

Line 4 in Fig. 4, (Forbid (Equal 1 (Find carbons attached-to C3))), is an alternate method for disallowing the formation of primary ions. This command illustrates that the language is context sensitive; a *Find* expression within a *Label-site* command will return all topologically unique items. Implicit in Line 4 of Fig. 4, however, is that *Find* will locate and count all of the carbon atoms adjacent to the atom labeled C3. Because the *Find* expression is not located in a *Label-site* command, the carbon atoms may be connected to C3 by a bond of any order.

THE COMPILER

The compiler translates the reaction descriptions into the internal representation of the generic reactions that is used by the network generator. The modules of the RDL compiler are shown in Fig. 5.

The parser verifies that the input is syntactically correct and forms an intermediate representation of the reaction. Each command is analyzed through the production rules of the nonterminals. One part of the parser analyzes expressions beginning with preposition operators (e.g., attached-to, belonging-to) while another part analyzes manipulation statements. Each word in an RDL statement is replaced by the appropriate token, depending on its context; for example, a token representing a *Find* operator located within a *Label-site* statement would be modified to reflect that only unique items (atoms or bonds) should be located.

```
    {
1 (Label-site Cl+ (Find positive-carbon))
2 (Label-site C2 (Find neutral-carbon attached-to C1+))
3 (Label-site C3 (Find neutral-carbon attached-to C2))
4 (Forbid (Equal 1 (Find carbons attached-to C3)))
5 (Disconnect C2 C3)
6 (Increase-order-of bond connecting C1+ C2)
7 (Subtract-charge C 1+)
8 (Add-charge C3)
9 (Set Number-of-reactions (Symmetry-number-of C3))
    }
```

Fig. 4. A set of commands used to specify β-scission reactions.

The command (Label-site C2 (Find positive-carbon attached-to C 1)) is recognized as an assignment statement after the first word (Label-site) is read; the resulting parse tree is shown in Fig. 5. According to the production rule for an assignment statement, the second parameter must be a user-defined variable, while the remainder of the statement must be an item expression: a variable, an item descriptor, or a *Find* expression. The parser examines the second and third parameters of the statement accordingly. An error is signaled if there are more than two parameters after an assignment operator. If the variable C2 in statement (12) has been previously defined, a warning is issued. The third parameter of the statement, an item expression, is recognized as a *Find* expression; it is sent to the part of the parser that analyzes *Find* expressions. Because the expression is within a *Label-site* command, the token for the operator *Find* will be changed to indicate that only topologically unique items should be located. Assuming that Cl represents an atom, the token *attached-to* will be changed so that only positive carbons connected to C 1 by a single bond will be located. If the variable C 1 has not been defined, an error is issued.

In addition to syntactic analysis, the parser also carries out semantic analysis, which is facilitated by the highly specialized vocabulary of the language. The expressions and statements of the language are organized into sub-groups based on what they ultimately return at run time. For example, the production rule for an arithmetic statement is:

$$\text{arithmetic-stmt} \rightarrow (\text{arithmetic-op number-expr*}) \quad (13)$$

where number-expr* is one or more expressions th: t return numbers. However, in our language there are only five possible types of expressions that result in a number: a slot expression, another arithmetic statement, a find expression, a test statement, or a set of characters that only contain digits. These expressions are defined formally as number expressions. Therefore, if another type of expression follows an arithmetic operator, an error is issued at compile time.

The parser interacts closely with the lexical analyzer, which reads the individual words and creates the appropriate tokens. The words of RDL may not contain white space (i.e., a space, tab, or new line character) and they must be separated from each other by white space. Most words can be identified and transformed into the appropriate token by a simple lookup call to the symbol table. If a word is either an operator, unmodified item, previously-defined variable, or a previously encountered modified item (an item with a prefix and/or suffix) it is located in the symbol table. The main function of the lexical analyzer, therefore, is to distinguish between user-defined variables and legally modified items (e.g., "negative-oxygens" is legal whereas "negative-bonds" is not). When an unknown word is encountered, the lexical analyzer first determines whether or not the word contains a hyphen, which indicates that the word may be an item descriptor modified by a prefix. If a hyphen is found, the characters before the hyphen are collected and a lookup call to the symbol table is made to determine whether the characters represent a prefix. The remaining characters are then analyzed to determine whether they represent an item (with or without a suffix). The prefix and item, if found, are checked for compatibility; if they represent a legal modified item, the appropriate token is created and added to the symbol table. Suffixes are treated in a similar manner. If the word represents a new user-defined variable; an identifier token, representing the variable, is created, added to the symbol table, and returned to the parser.

After the entire reaction description is parsed, it is sent to the optimizer. Currently, this module marks the statements of a reaction as belonging to one or more categories. This information is used in the translation and in simple optimization. The two main categories are the pretransform commands and the remaining commands. Pretransform commands include all *Label-site* statements and any statements that occur prior to the first manipulation statement. The number and type of variables referenced by each command is determined. A command that only refers either to one variable or to no variables is defined as a simple command. A global command is a type of simple command where the single variable, if present, represents the reactant, product, or reaction as a whole. The command (Disconnect Cl hydrogen) is a simple command, whereas (Require neutral reactant) is a global command. The commands are further partitioned based on the first word of the command; for example, a command beginning with the operator *Require is* a pruning command.

Pretransform pruning commands are placed at the beginning of the reaction specification. It is inefficient to locate numerous reaction sites in a compound only to discard them due to global features of the compound. When finding the reaction sites for hydride abstraction, it is undesirable to locate all neutral carbon-hydrogen pairs in a carbenium ion, only to discard them because of a command that requires that the reactant be neutral.

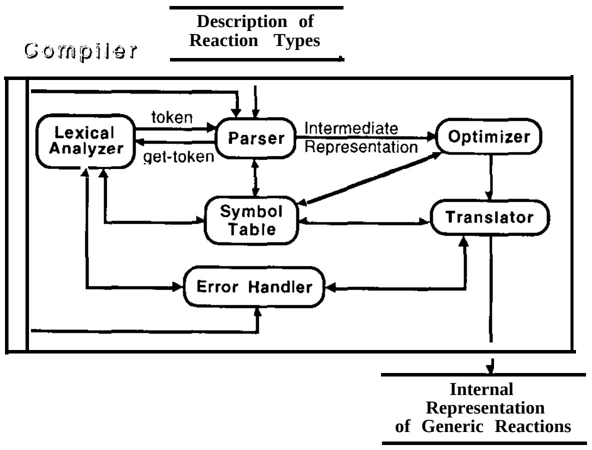
Compiler
Description of Reaction Types
token
get-token
Lexical Analyzer
Parser
Intermediate Representation
Optimizer
Symbol Table
Translator
Error Handler
Internal Representation of Generic Reactions

Fig. 5. The modules of the RDL compiler.

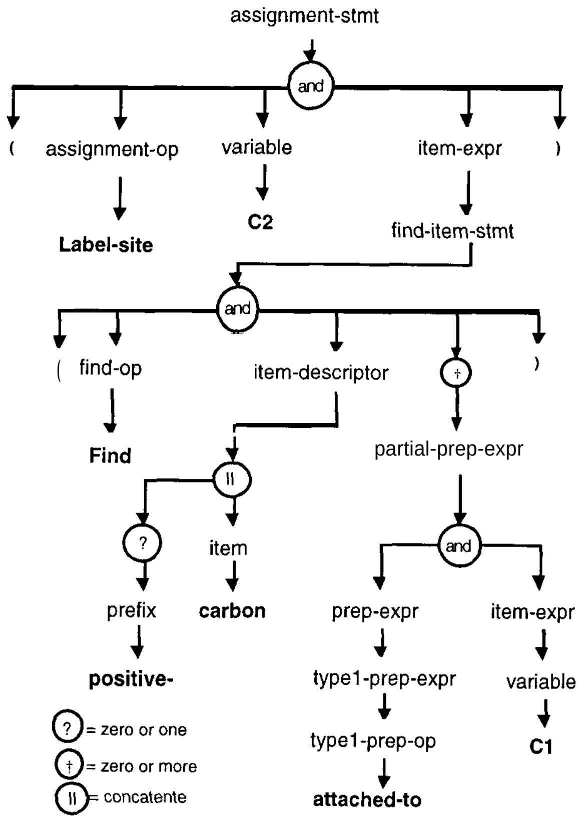

Fig. 6. The parse tree of the statement (Label-site C2 (Find positive-carbon attached-to Cl)).

Currently additional optimization techniques are being developed that affect both the compile time and run time environments. One optimization technique uses an information vector, based on the search and pruning commands. The vector may include, for example, the compounds total charge, number of multiple bonds, and other functional groups. The information vector acts as a filter; only compounds that conform to the vector specifications are further analyzed. Thus, if a command was included to locate a ring, only reactants that contained rings would be analyzed. These techniques are transparent to the user, who need not consider the optimization when writing a reaction specification.

The marked rearranged commands are sent to the translator for the final formation of executable reaction functions, used in the generation of the reaction network.

CONCLUDING REMARKS

A new approach for the computational generation of complex reaction systems, based on a language for describing the general type of reaction steps has been developed. Due to the large number of reactants, intermediates, and reactions involved, there are difficulties in modeling or even describing complex systems. The Reaction Description Language can be used to generate and simplify any type of reaction system; the language provides a method for describing reactions that is not based on a specific class of reaction types. It allows considerable flexibility in the definition and modification of reactions, and is helpful in limiting the complexity of a reaction network. The RDL compiler and network generator have been tested on systems with several thousands reactions and hundreds of intermediates.

Related methods occur in Computer Aided Organic Synthesis (CAOS). The broad range of modeling strategies in CAOS is discussed by Ugi *et al. (1994).* Methods can be classified as information-oriented or logic-oriented (Ott and Noordik, 1992). Information-oriented systems such as LHASA (Pensak and Corey, 1977) rely on a library of well-understood reactions. Logic-oriented programs, such as EROS (Gasteiger *et al.,* 1990), SYNCHEM (Gelernter *et al.,* 1973), and CAMEO (Jorgensen et al., 1990), rely on a mathematical model of constitutional chemistry such as that developed by Dugundi and Ugi (1973). These systems are often used to find unprecedented reactions. Both approaches include a set of constraints/heuristics to evaluate the applicability of a reaction. Many systems rely on the user to ultimately acceptor reject a particular pathway. All strategies must, however, contend with the combinatorial explosion which arises from the large number of potential reactions that may be chosen at any particular point in the generation of a reaction pathway.

In the RDL system presented here, reactions are described using a sequence of commands to characterize the reaction site, the transformation of the reactants to products, and optional pruning constraints. These reaction descriptions are compiled and used by a network generator.

Auxiliary components are needed to transform the input representation of the molecules into a form that may be manipulated by the network generator. Simple text strings are used to specify the individual molecules. These reactants, along with the subsequently generated intermediates, must then be analyzed: This involves determining the important structural features of the molecule, such as the presence of aromatic rings, as well as determining a unique canonical representation of the compound. Canonicalization allows efficient determination of the equivalence of two or more compounds or atoms.

The RDL compiler and network generator have been used to generate complex systems involving thousands of specific reactions and hundreds of intermediates (Prickett, 1994), and auxiliary methods have been developed to analyze and simplify these networks.

REFERENCES

Mavrovouniotis, M. L., Constantinou, L., and Prickett, S. "Computer-Aided Analysis of Ionic Reaction Systems." *Proceedings of IFAC Symposium on Dynamics and Control of Reactors, Distillation Columns, and Batch Processes (DYCORD+92)*, Pergamon Press, 1993.

Prickett, S.E. "Object-Oriented Generation of Complex Reaction Systems for Chemical Processes. " Ph.D. thesis (advisor: M.L. Mavrovouniotis), University of Maryland, College Park, 1995.

Agnihotri, R. B.; Motard, R.L. "Reaction Path Synthesis in Industrial Chemistry", *A CS Symposium Series 124*, ACS: Washington, D.C. (1980).

Dugundi, J.; Ugi, I. "An Algebraic Model of Constitutional Chemistry as a Basis for Chemical Computer Programs". *Topics Curr.Chem. 39, 19-64 (1973).*

Gasteiger, J.; Ihlenfeldt, W. D.; Rose, P.; Wanke, R. "Computer-assisted reaction prediction and synthesis design". *Anal. Chim. Acts 235, 65-75 (1990).*

Gelernter, H.; Sridharan, N. S.; Hart, A. J.; Yen, S. C.; Fowler, F. W.; Shue, H.-J. "Discovery of Organic Synthetic Routes". *Topics Curr.Chem.* **41, 113-150 (1973).**

Jorgensen, W. L.; Laird, E. R.; Gushurst, A. J.; Fleischer, J. M.; Gothe, S. A.; Helson, H. E.; Paderes, G. D.; Sinclair, S. "CAMEO: a program for the logical prediction of the products of organic reactions". - *Pure & Appl.Chem.* **62, 1921-1932 (1990).**

Ott, M. A.; Noordick, J.H. "Computer tools for reaction retrieval and synthesis planning in organic chemistry. A brief review of their history, methods, and programs". *Reel. Trav.Chim.Pays-Bas* **111, 239-246 (1992).**

Pensak, D. A.; Corey, E.J. "LHASA - Logic and Heuristics Applied to Synthetic Analysis" in *Computer-Assisted Organic Synthesis,* W.T. Wipke and W.J. Howe, eds., ACS Symposium Series 61, ACS: Washington, D. C. (1977).

Ugi, I.; Bauer, J,; Blomberger, C.; Brandt, J.; Dietz, A.; Fontain, E.; Gruber, B.; ScholleyPfab, A.; Senff, A.; Stein, N. "Models, Concepts, Theories, and Formal Languages in Chemistry and Their Use as a Basis for Computer Assistance in Chemistry". *J.Chem.Inf. Comput. Sci. 34, 3-16* **(1994).**

*****

Suzanne E. Prickett

Suzanne Prickett received her Ph.D. in Chemical Engineering at the University of Maryland, College Park, in 1995. She is the recipient of a Systems Research Fellowship and a Master's Thesis Award from the College Park chapter of the American Association for University Women. Her research interests lie in the general area of computer-aided process engineering and process development. She can be contacted at 2836 Winchester Way, Falls Church, VA 22042.

Michael L. Mavrovouniotis

Mike Mavrovouniotis is an Associate Professor of Chemical Engineering at Northwestern University. He received his Ph.D. from the Massachusetts Institute of Technology in 1989. His recent teaching has included Process Design, Statistics, and Separations. His research interests lie in the general area of computer-aided engineering of chemical processes, including process design, analysis, modeling, and monitoring. He is the recipient of Best Paper awards from the journal Computers and Chemical Engineering for 1987 and 1992, and the 1991 Ted Peterson award of the Computing and Systems Technology division of AIChE. He is the ASEE Campus Representative at Northwestern. He can be contacted at (708) 491-7043, fax 708 491-3728, by email at mlmavro@nwu.edu, or at the Chemical Engineering Department, Northwestern University, Evanston, IL 60208-3120.