MODELING LANGUAGES: DECLARATIVE AND IMPERATIVE DESCRIPTIONS OF CHEMICAL REACTIONS AND PROCESSING SYSTEMS

Christopher J. Nagel, Chonghun Han, and George Stephanopoulos

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

To model is to represent reality, and modeling as an essential task of any engineering activity is always *contextual*. Within the scope of differing engineering contexts, the same physical entity, such as a molecule, chemical reaction, or process flowsheet, is represented with a broad variety of models. An enormous amount of effort is expended in the development and maintenance of a *Babel of models*, sporting different languages and being at cross-purposes with each other, although like their biblical counterparts share a common progenitor: in this case, the fundamentals of chemistry and physics and the principles of chemical engineering science. Creating a language that supports the expeditious generation of consistent models has become the key to unlocking the power of many computer-aided tools, and unleashing the explosive synergism between human and computer. However, a modeling language is of little use if it only creates representations of physical entities as "things onto themselves" without meaningful semantic designation to what it purports to represent. Furthermore, the model of an entity *should contain all knowledge* that has some bearing to the representation of that entity, whether that is *declarative* or *imperative* (procedural) in character. In this chapter we will describe two modeling languages: LCR (Language for Chemical Reasoning) to represent molecules and chemically reactive systems, and MODEL.LA. (MODELing LAnguage) for the representation of processing systems. Both are based on the same principles and have, to a large extent, a common structure. Both have been based on ideas and techniques, which originated in artificial intelligence, and both have been implemented in a similar object-oriented programming environment.

I. Introduction

Since the early efforts, computer-aided modeling of physical systems has been generally organized around unilateral computations that perform predetermined operations on fixed inputs to produce the desired outputs. This is still a common practice since it helps engineers and scientists to manage tedious calculations and coordinate and control diverse numerical tasks, by producing models that use very efficiently computer resources. However, a series of inherent weaknesses have pointed out the disadvantages of the traditional approach: (1) the time and cost associated with computer model development are high; (2) the resulting models are difficult to document and maintain adequately; (3) the reuse of computer-aided models is minimal, as these models tend to be task-specific and are often intrinsically linked to solution procedures; (4) the models cannot be synthesized automatically by the computer in the course of automatic execution of an engineering task; and (5) for interactive modeling the modeler is required to be highly skilled in programming. As a result of these weaknesses the duplication of modeling efforts has been enormous. Accumulated modeling knowledge is almost impossible to use, since the underlying modeling context (purpose, assumptions, simplifications) has never been documented and rationalized. So, why must every new modeling effort start from scratch (Meyer, 1987). Furthermore, automatic generation of models at higher abstractions, cannot be done. Finally, the fragmentation of modeling efforts and the ad hoc character of their computer implementation have led to internal inconsistencies among the various models used in different process engineering tasks. A typical example of this phenomenon is found in the diversity of models used to support process-control-related engineering tasks (Stephanopoulos, 1990) such as design of process controllers, controller adaptation mechanisms, optimization of process operations, and diagnosis of process faults.

A. The Five Premises of a Modeling System

To overcome the weaknesses discussed above, any modeling system should be built on the following premises.

1. Articulate All Declarative Knowledge

The model of an entity, whether a processing unit, a molecule, or a chemical reaction, should contain explicitly all relevant information, including the following (Stephanopoulos *et al.*, 1990a):

(a) *Underlying assumptions.* The form of the relationships that describe the behavior of a given entity depends on a series of assumptions, such as the operational mode; the assumed mechanisms for mass, energy, or momentum transfer; the mechanistic pathway for a chemical reaction; and the constitutive models for the estimation of physical properties.

(b) *Simplifications* made by the modeler (human, or another computer program) in order to limit the model's validity over a given range of conditions, or to underscore the relative importance of various physicochemical phenomena.

(c) *Scope of engineering task*, i.e., what the model is intended for. Typical examples are (1) the variety of models required by the different levels of the hierarchical synthesis of process flowsheets (Stephanopoulos, 1990a, 1990b); (2) the variety of models needed for feedback and adaptive control, diagnosis, or planning of process operations (see the fifth chapter in this volume).

(d) *Relationships among the various variables and parameters*. These relationships express the fundamental principles of physics and chemistry as well as the assumed mechanisms for rate and equilibrium phenomena, and the correlations for the estimation of physical properties. These relationships could be quantitative, qualitative, or semiquantitative (e.g., order-of-magnitude, or ordinal), depending on the level of the available knowledge, and could express static or dynamic behavior.

2. *Separate Declarative from Procedural Knowledge*

The declarative knowledge that is articulated in a given model represents the "what is" knowledge about the modeled entity. Since the same declarative model could be used for a variety of engineering tasks, it is essential that it be separated from the procedural knowledge, i.e., the "how to" knowledge of a particular engineering methodology. Previous modeling approaches relied on a tight integration of declarative and procedural knowledge with the sequential modular simulators (e.g., ASPEN; see Evans *et al*., 1979) exemplifying the tightest integration, and the equation-oriented simulators (e.g., SPEEDUP; Perkins and Sargent, 1982; Pantelides, 1988) offering a weaker but still dominant integration.

3. *Hierarchical and Multiview Representation of Entities*

A modeling system should allow the representation of the various entities at any level of detail, using multiple, coexisting abstractions, which can communicate with each other. This is a critical requirement as it sets the desired modeling system apart from many other modeling environments. For example, a processing system could be represented in any of the following abstractions (Fig. 1): (a) an overall plant, (b) a network with generalized reaction and separation sections, (c) a network of reactors and abstract separation sections, or (d) a network of processing units. All models, depicting the four abstraction views in Fig. 1, should be consistent with each other and should allow the transfer of information from more abstract to more detailed representations and vice versa. In addition, two distinct and different versions of the same abstraction (e.g., Fig. 2) have many modeling components in common. The modeling system should allow common handling of the common modeling elements. For example, the modeling relations for the generalized reaction and separation sections

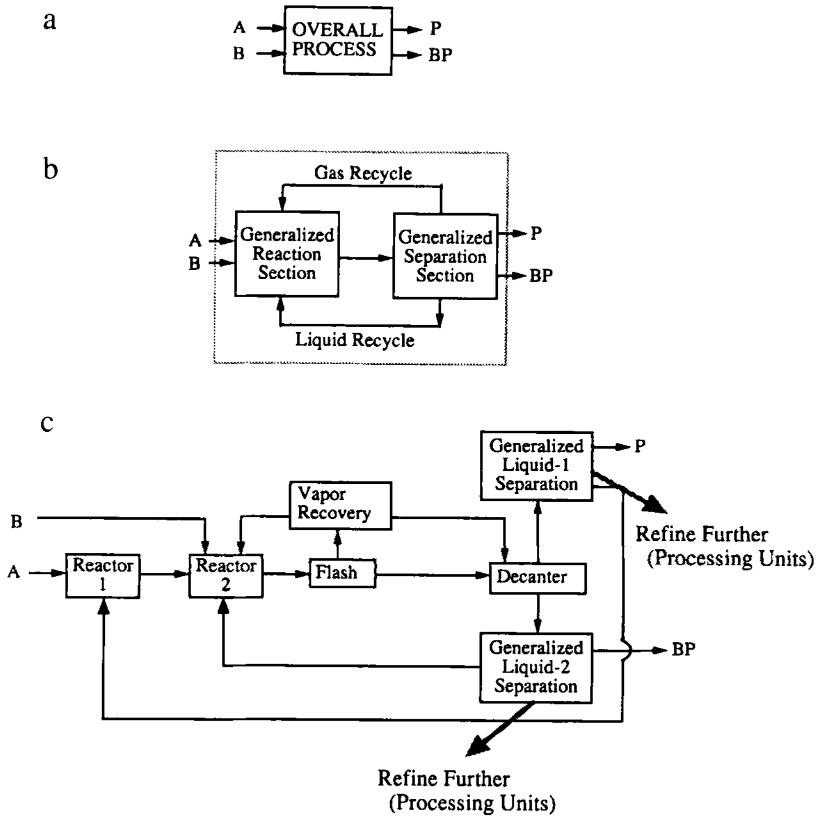

a
A
B
OVERALL PROCESS
P
BP
b
Gas Recycle
Generalized Reaction Section
Generalized Separation Section
Liquid Recycle
c
Reactor 1
Reactor 2
Vapor Recovery
Flash
Decanter
Generalized Liquid-1 Separation
Generalized Liquid-2 Separation
Refine Further (Processing Units)
Refine Further (Processing Units)

FIG. 1. Three abstraction levels for the representation of a processing scheme.

of the two versions in Fig. 2 are the same. If we change the models in one of the versions, the representation of the other version should be updated automatically.

4. *Automatic Generation and Modification of Models*

Once the modeler has described the elements of the modeling premise 1—i.e., made the underlying assumptions and simplifications and defined the scope of the engineering task—the modeling system should automatically generate the modeling relationships and their elements. In other words, the modeling language should possess declarative representations of relationships, of the terms that made up such relationships, of the variables that make up the terms, and of the semantic connections that

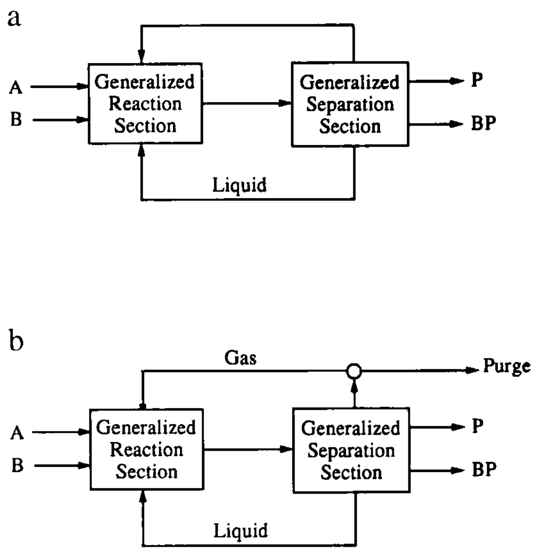

a
A
B
Generalized Reaction Section
Generalized Separation Section
P
BP
Liquid
b
Gas
Purge
A
B
Generalized Reaction Section
Generalized Separation Section
P
BP
Liquid

FIG. 2. Two distinct versions of a plant at the same abstraction.

assign physical meaning to the variables, terms, and relationships. Furthermore, once the modeling system has developed the model of an entity, it should be capable of updating the model automatically, if the modeling assumptions and simplifications have changed.

5. *Simulation and Reasoning*

The modeling relationships of the various entities should possess enough granularity to allow structural reorganization for efficient numerical simulation and logical reasoning. For example, given the static quantitative model of a process, the modeling system should possess the necessary modeling information to allow the construction of the corresponding cause and effect, directed graph, or automatically analyze the degrees of freedom and generate the simplest numerical procedures for simulation.

The preceding five premises imply requirements that go beyond the scope of traditional computer-aided modeling systems. The modeling language should allow interaction with the modeler at a high level of abstraction, while the generated models should be explicit, transparent, fully documented, and usable without the need to consult manuals for assumptions or input/ output language conventions.

B. Review of Modeling Systems for Process Simulation

In recent years several modeling systems have appeared in the literature, all of them aimed at eliminating the perceived modeling bottleneck in generating static or/ and dynamic simulations of processing systems. Consequently, all the modeling systems to be discussed in the subsequent paragraphs do not conform to all of the five premises, discussed in the previous section, and consequently they cannot be used outside the simulation tasks that they were designed to serve.

1. ASCEND (Advanced System for Computations in ENgineering Design)

As its name implies, ASCEND is a general computer-aided environment with a modeling language that was developed to support the expeditious generation of mathematical models that are needed for the simulation and design of processing systems (Piela, 1989; Piela *et al.*, 1991). It follows an object-oriented paradigm, but it is strongly typed. Its main features can be summarized as follows:

a. Building Blocks. ASCEND uses three types of generic modeling elements: `model`, `atom`, and `type`. Models are structured entities that are built hierarchically from (1) instances of other models, atoms, or/ and types; and (2) relationships between these instances. Atoms are primitive variables, which are used to represent physical quantities, design variables, etc. Types are elementary declarations of real, integer, string, Boolean, etc. They are predefined in ASCEND.

b. Relationships. Several operators define relationships among the various elements of ASCEND. Three of them are used to define inheritance relationships: (1) the ***refines*** operator is used to link a subclass to a class in an inheritance hierarchy; (2) the ***is-a*** operator is used to declare an object as the instance of a class (or, a type); and (3) the ***is-refined-to*** operator changes the type associated with a previously declared instance. Two relationships are used to group instances together: (1) the ***are-alike*** operator coerces all members of a group of instances to take the type associated with the most refined instance; (2) the ***are-the-same*** operators is similar to the previous one, but merges the operands into a single structure.

c. General Features. ASCEND uses the inherited mechanism of the object-oriented paradigm in order to provide expeditious extension of its "vocabulary." It contains both instances and classes, but instances are not allowed to have local attributes. ASCEND generates mathematical relationships that are expressed as implicit equations or solved assignments. It uses a fairly concise syntax to express a fairly broad set of modeling statements.

2. *OMOLA (Object-oriented MOdeling LAnguage)*

OMOLA (Andersson, 1989; Nilsson, 1993) was designed to support the model generation and simulation needs during the computer-aided synthesis and analysis of control systems. It is oriented toward the creation of structured dynamic systems. Its main features can be summarized as follows:

a. Modeling Elements. All models are created as subclasses of the mother class, `Model`, thus inheriting all the attributes of the mother class. It is possible to create specialized versions of an existing model, M1, by creating a subclass from the class, M1. The description of each model is organized around four modeling elements; `Terminal`, `Parameter`, `Variable`, and `Realization`. The first is used as a point of communication between the models of the various entities. `Parameter` and `Variable` are used to describe model attributes that are time-independent and time-varying, respectively. `Realization` is the element that is used to define the mathematical relationships describing the behavior of an entity. Each of the four modeling elements is described by a structured class with a series of attributes. Specialized subclasses emanate from `Terminal` and `Realization`, thus allowing an expansion in the vocabulary of the language itself.

b. Functionality. OMOLA allows the representation of complex structured systems through the aggregation of individual models. The language offers no semantic relationships to link the generated models or their attributes to any physical concepts. Consequently, it cannot be used to automate the generation of models from a set of physical modeling assumptions. Also, it does not allow for the articulation of qualitative or semiquantitative knowledge.

3. *MODASS (MODeling ASSistant)*

MODASS (Sørlie, 1990) is more of a toolbox for the construction of models. Although it allows declarative description of models and their elements, it also permits the incorporation of subroutines as model components.

a. Modeling Elements. The modeling elements of MODASS are assembled in a hierarchy of classes. At the top is the class, `model`, containing very few common attributes. At the next level we encounter the subclass, `Model-Element`, which contains the basic building blocks of the actual model. Further specializations (i.e., subclasses) provide narrow specifications of the sets of equations used for modeling specific entities. To represent the definitional boundary of an entity and the gateways for the information exchange between different entities, MODASS uses the classes, `Process-Terminal` and `Boundary-Process`, respectively. Specializations of the latter class allow the tailoring of the modeling element to mass, energy, and information transfer.

b. Functionality. MODASS has very limited network of semantic relationships among the different modeling elements, but it offers a satisfactory set of tools to the user for (1) the navigation through the tree of library models, (2) the symbolic manipulation of the modeling equations, and (3) the examination of models for inconsistencies and errors. Its lack of meaningful semantic relationships among the various modeling elements and the physical entities they represent makes MODASS a modeling system with limited scope, and largely outside the framework of the five premises, discussed in Section I.A.

4. Hybrid Phenomena Theory

Hybrid phenomena theory (HPT) (Woods, 1993) is not a modeling language in strict terms, but a theoretical framework for the generation of models that capture the topological, phenomenological, and behavioral aspects of the entity being modeled. The *topological* model of an entity includes a set of objects, (e.g., pipes, vessels, valves) and a set of logical relationships among the objects. The *phenomenological* model of the entity contains a set of objects, describing instances of physical phenomena that occur in the system being modeled. These phenomena are active processes such as heat transfer, material accumulation, and convective flow. The *behavioral* model is a state-space model providing a quantitative description of the entity's dynamic (or, static) behavior. The topological, phenomenological, and behavioral modeling components of an entity are tightly integrated and information flows among them as needed.

a. Modeling Elements. The object-oriented character of HPT places the `quantity` as the mother-class of the subclasses `constant`, `parameter`, and `variable`, whose further specialization has produced the subclass `state-variable`. A second hierarchy of classes, emanating from the mother-class `object`, captures the information available for various physical objects, such as materials (represented by class `stuff` and its specializations), processing units (from class `process-equipment`), devices (from class `device`). A hierarchy of `conditions` provides the logical mechanism for monitoring the values of data and taking action when specific conditions are met.

b. Functionality. The modeling elements, described above, are used as the building blocks for the construction of higher-level modeling objects, such as `views` and `phenomena`, which describe physical interactions. The topological model of HPT provides a Boolean representation of the model's components, whereas the phenomenological model supplies a qualitative description of the entity's behavior. It is the behavioral, state–space model that carries the full quantitative description.

5. *Critique of Modeling Systems for Simulation*

All the systems, described in the previous paragraphs, were created with a very specific purpose in mind: to expedite the generation and modification of quantitative models to support the needs of static or dynamic simulation or design. As a result, they are very difficult (if possible at all) to be used for other engineering tasks, which may have a different use for the generated models (e.g., diagnosis). Their common weakness results from two essential deficiencies: first, they cannot capture all forms of knowledge (e.g., qualitative or semiquantitative relationships); second, they do not possess a rich set of semantic relations, which allow intelligent response to queries about (a) the assumptions that produced a given model, (b) the interrelationship of physical phenomena captured in the modeling relations, (c) the propagation of information flow among different views of the same entity, etc. In Section IV we will see how the modeling language MODEL.LA. can overcome these deficiencies.

C. Modeling Systems in Chemistry

With the exception of computational chemistry, computer-aided chemical reasoning has not enjoyed the success experienced in other scientific disciplines. This has often been attributed to the conceptual nature of chemistry and the inexactness of known relationships (Dugundji and Ugi, 1973). However, the synthesis of new chemical reaction paths and processing alternatives that support their production remains an attractive objective within the general field of process synthesis. The availability of raw materials and energy, changing ecological and health considerations, and shifting requirements of the market, reinforce each other: to create the need to identify new routes or new processes to obtain existing chemicals or develop new chemicals to meet perceived requirements.

Chemists were the first to look into computer-aided organic synthesis (CAOS) in an attempt to answer Woodward's question (Woodward, 1956, 1963) and furthered by Corey (1967): "How does a chemist choose a pathway for the synthesis of a large organic molecule, given either the great diversity of organic structures and reactions, or, in contrast, the critical importance of each step to ultimate success?" The works of Corey, Wipke, Ugi, Hendrickson, Jorgensen, Moreau, Gelernter, and others support the CAOS effort (see Table I). Excellent reviews can be found in the works of Vernin and Chanon (1986) and Wipke *et al.* (1974).

For any problem of computer-aided organic synthesis, the quality and quantity of required data will vary, the most appropriate strategies may be different, but the questions central to the advancement of computer-aided chemical reasoning remain the same:

(a) How should the molecules and reactions be represented?
(b) What is the necessary degree of descriptional completeness, and at what cost can it be achieved?

Just as Lavoisier stated 200 years ago, "It is time to rid chemistry of obstacles of every kind... this reform must be brought about by perfecting the language," we postulate that a high-level (computer) language must be developed for meaningful advancement to occur in computer-aided chemical reasoning.

Domain-specific modeling languages are "very high level" and of "special purpose." They have a *theme*, that is, the class of ideas that is optimized to communicate. They allow the user to employ terms and constructs that lie closer to the informal terminology and modes of speech customary in the discussion of domain-specific problems; however, in constructing such a language, one should strike for domain-specific generality of the language. Thus, the same modeling language should satisfy all the needs: chemical synthesis, reaction analysis, pathway optimization, and innovative design of reaction or processing networks. The implication of this requirement is clear: *A modeling language in chemical reasoning should be fully declarative, and in no way should its generality be compromised by the specificity of the methodologies of the chemical tasks themselves.*

TABLE I
COMPUTER AIDS IN CHEMISTRY

Number	Authors	Program name	First published
1	Corey-Wipke	OCSS	1969
2	Corey	LHASA	1971
3	Ugi-Gasteiger	—	1971
4	Hendrickson	—	1971
5	Bersohn	—	1971
6	Weise	AHMOS	1973
7	Gelernter	SYNCHEM	1973
8	Barone-Chanon	SOS	1973
9	Powers	DINASYN	1973
10	Brownscombe	EXTRUS	1973
11	Brownscombe	HEXARR	1973
12	Wipke	SECS	1974
13	Ugi-Gasteiger	CICLOPS	1974
14	Benedek	SIMUL	1974
15	Dubois	SYNOPSYS	1975
16	Whitlock	—	1976
17	Donova	HEDOS	1976
18	Powers	REACT	1977
19	Govind	REPAS	1977
20	Djerassi	REACT	1977
21	Pensak	LHASA (Du Pont)	1977
22	Kaufmann	PASCOP	1978
23	Moreau	MASSO	1978
24	Gelernter	SYNCHEM2	1978
25	Ugi-Gasteiger	EROS	1978
26	Yoneda	GRACE	1978
27	Gasteiger	PSYCHE	1979
28	Weise	GSS	1979
29	Barone-Chanon	SAS	1979
30	Stolow-Joncas	LHASA (Educ)	1980
31	Agnihotri	CHIRP	1980
32	Jorgensen	CAMEO	1980
33	Gund	SECS (Merck)	1980
34	Hendrickson	SYNGEN	1981
35	Hippe	SCANSYNTH	1981
36	Hippe	SCANPHARM	
37	Hippe	SCANMAT	
38	Zefirov	FLAMINGOES	1981
39	Ghose	—	1981
40	Schubert	ASSOR	1981
41	Zin	—	1982
42	Erdos	ASR	1983
43	Kaufmann	PSYCHO	1984
44	Seidel	—	1984
45	Hara	PFP	1984
46	Wipke	SST	1984
47	Cense	MICROSYNTHESE	1985
48	Barone-Chanon	TAMREAC	1985

In the absence of any formal modeling language for the representation of chemicals and chemical reactions, it is important to state a concise set of requirements that the desired language should satisfy. These requirements, consistent with the five premises of Section I.A are

1. Represent molecules, reactions, and reaction pathways at various levels of detail.
2. Allow easy extension to new classes of molecules and reactions.
3. Allow the contextual representation of chemicals' reactivity, i.e., reactivity influenced by the relative position of atoms, the surrounding conditions and presence or absence of specific molecules.
4. Support all scientific or empirical knowledge brought together during the synthesis of reaction pathways.

In Section III we will see how LCR (Language for Chemical Reasoning) meets these requirements.

II. LCR: A Language for Chemical Reasoning

In this section we will describe the components of a modeling language, called LCR, that was developed to represent the knowledge about chemically reacting systems. First, we will describe the *basic modeling elements*, which constitute the building blocks for the representation of the declarative knowledge about molecules, reactions, and pathways. Second, we will discuss the semantic relationships among the basic modeling elements. These semantic relationships establish the "meaning" behind the linguistic expressions, defining knowledge about molecules and reactions. Third, we will present the syntax used by the language for the description of chemically reacting systems. LCR was implemented on a Symbolics 3650. It consists of approximately 50,000 lines of LISP (excluding code for the interfaces). Extensive discussion on the use of LCR for pathway generation can be found in Nagel (1991). The fourth chapter in this volume also discusses how LCR has been used to generate the reaction-based potential hazardous events in a chemical plant.

A. Modeling Elements of LCR

A single chemical structure may exhibit multiple chemical behaviors, depending on the conditions in the surrounding environment. As a result, LCR uses two groups of modeling elements; the first provides those elements that are needed to describe the structure of the molecule (e.g., atoms, bonds, connectivity), and the second class contains those elements that characterize its chemical behavior and are related to the atom's electronic configuration.

1. Modeling Elements Defining Chemical Structures

LCR uses a graph theoretic representation of molecular structures, wherein nodes are atoms and edges are bonds. Three modeling elements are employed to create and describe chemical structures. Each modeling element has been implemented as a *class*, in an object-oriented programming environment, and is described by a set of attributes and a set of procedures.

a. Modeling Element 1: `atom`. An atom is the building block of chemical structures? Each `atom` encapsulates its own structural information, modeling relationships, and modeling assumptions. The attributes describing the object class, `atom`, are shown in Table II. Additional attributes may be associated with the `atom` class to refine the class description. For example, the attributes, formal-charge, oxidation-state, steric-hindrance, chirality, may be included into an atom's description to refine the class description. Several of the attributes given in Table II warrant further explanation: Attributes "identifier" and "old-identifier" are pointers used to trace an atom's lineage; attribute "database-atom" contains an object `db-atom` that manages invariant information indigenous to a particular atomic species (e.g., atomic-symbol, atomic-number, atomic-weight). The attribute "parent-abc" associates an atom to a given chemical structure; its value provides the vehicle for moving between the atom representation and the "parent-abc" representation. "Parent-group" is used similarly. An `atom` may have an association with a functional group or set of groups, as well as a parent-abc. As in the attribute case of "parent-abc," the value of "parent-group" provides a conduit for accessing information residing at the `group` representational level. As will be shown later, all representational levels can be accessed from any individual level. The freedom of information flow between levels enhances representational expressiveness and facilitates efficient reasoning.

A partial listing of the methods operating on the class `atom` is also shown in Table II. Methods with the "compute" prefix (e.g., *compute-parent-groups*, *compute-hybridization*, and *compute-formal-charge*) are used to evaluate attribute values. Predicate methods (i.e., methods returning a Boolean value) are represented by the "-p" suffix. These methods are

TABLE II

SELECT ATTRIBUTES AND METHODS FOR `atom`, `bond`, AND `atom-bond-configuration` CLASSES

`atom`	`bond`	`atom-bond-configuration`
atom attributes	**bond attributes**	**atom-bond-configuration attributes**
identifier	identifier	identifier
old-identifier	character	atoms
name	strength	bonds
type	length	empirical-formula
chiral-p	type	molecular-weight
formal-charge	atoms	charge
electronegativity	parent-abc	equivalent-atoms
electron-withdrawing-	progenitors	equivalent-bonds
substituent-pacidity	db-bond	weakest-bond
connectivity	**bond-methods**	heat-of-formation
hybridization	methods	entropy-of-formation
conjugated-p	find-bond-chains	free-energy-of-formation
p-orbitals	cleave-bond	homo
open-approach-p	compute-bond-strength	lumo
radical-orbital	remove-bond	progenitors
bond-angle	add-bond	environment
parent-abc	modify-bond	...
progenitors	create-bond	**atom-bond-configuration-methods**
parent-groups	**bond-selectors**	setup-atom-bond-graph-descriptor
neighbor-atoms	same-bond-type-p	atom-bond-graph-descriptor
neighbor-groups	bond-equivalence-p	setup-atom-bond-graph-from-old-graph
bonds	alpha-bonds	setup-atom-bond-configuration
database-atom	beta-bonds	make-graph-from-symmetric-
atom methods	gamma-bonds	connectivity-matrix
compute-connectivity-number	equivalent-bonds-p	make-bonds-from-connectivity-list
compute-parent-groups	terminal-bond-p	make-atom-bond-configuration
compute-neighbor-groups	internal-neighbor bonds	equivalent-atom-bond-configuration-p
compute-hybridization	terminal-bond-for-additions-p	make-old-arc-to-new-arc
compute-p-orbitals		identify-bond-printed-representation
compute-formal-charge		correct-bond-number-p
compute-radical-orbital		atom-symmetry-identification
identify-electron-withdrawing-		bond-symmetry-identification
substituent		enumerate-all-atom-chains
find-atom-chains		identify-atom-bond-configuration-
atom-backbone		environment
atom-degree		identify-atom-bond-configuration
higher-degree-atom-p		compute-equivalent-bonds
compute-open-approach-p		compute-weakest-bond
identify-conjugated-p		compute-equivalent-atoms
atom-specific-selections		...
create-atom		
abstract-grouping		
1,2-mobile-atom-p		
1,5-mobile-atom-p		
mobile-univalent-atom-p		

helpful when a characteristic of the atom is used by several different procedures, whether they are internal or external to the object class that contains them. For example, various operations may require knowledge about an atom's mobility as in the case of radical rearrangements; rearrangement alternatives are accessed using methods illustrated by *1,2-mobile-atom-p* and *1,5-mobile-atom-p*.

The remaining methods evaluate properties of an atom or properties of the parent structure. For example, *atom-backbone* identifies the skeleton of interest; *find-atom-chains* enumerates all the paths emanating from the specified atom; *abstract-atom-grouping* produces metagroups around reaction centers. Partitioning of the parent structure into active and inactive sites enables efficient manipulation during pathway construction, as we will see in Section III. New atoms are constructed using the method, *create-atom*. Selector functions, implemented as methods, are also built into the class `atom`. These methods provide an efficient means of collecting atom, which contain some specified desired property. Illustrative examples of selector methods include *collect-sp2-atoms*, *collect-oxygen-atoms*, *collect-beta-neighbors*, and *collect-terminal-atoms*.

b. Modeling Element 2: `bond`. Atoms are connected by bonds. Like atoms, each bond has associated with it structural information and models that manipulate this information to deduce new features that describe it. Bonds know the atoms that describe it. It is through these elements (bonds) that information is transferred from entity to entity, and new connectivity information is deduced.

A partial listing of the describing attributes and methods operating on object class `bond` is presented in Table II. Notice that the `atom`'s attribute "value" allows information flow from the `bond` representation to an `atom` representation, whereas the "parent-abc" attribute value allows movement between various levels of abstraction within the representation (e.g., groups and structures). In addition to evaluating attribute values, `bond` methods are used to facilitate the construction of chemical structures. To achieve this purpose, *cleave-bond*, *add-bond*, *remove-bond*, *modify-bond*, and *create-bond* are provided. Like atom selectors, bond selectors are used to collect bonds exhibiting a designated property to facilitate processing.

c. Modeling Element 3: `atom-bond-configuration`. Atoms and bonds containing connectivity information and spatial relations represent an instance of the `atom-bond-configuration` class. All chemical structures can be defined by an instance of the `atom-bond-`

configuration class. This structure, although in the strictest sense is not a primitive, has been elevated to a primitive to reduce complexity in subsequent reasoning. Associated with the atom-bond-configuration class are all methods and properties indigenous to an abstract chemical structure, such as physical and spatial properties. Thus, each specialized chemical structure is constructed from an atom-bond-configuration. An atom-bond-configuration in conjunction with the constituent atoms and bonds allows us to capture and isolate the structural features of a configuration from the feature that characterizes its behavior.

Generic chemical structures are represented by the object class atom-bond-configuration. These attributes describing this class include "name," "identifier," "atoms," "bonds," "empirical formula," and "frontier molecular orbital" (FMO). Since atoms and bonds are objects, the values of the attributes describing these entities are the set of objects constituting the atom list and the bond list, respectively. As a consequence, the values of the attributes describing these objects are easily accessible to procedures invoked by the atom-bond-configuration class. For example, the evaluation of an atom-bond-configuration's highest occupied molecular orbital (HOMO) requires evaluation of the atomic orbitals (AOs) that compose it. This information, which is resident in the atoms composing the atom-bond-configuration, is accessed through selector functions that are applied by the *compute-HOMO* procedure of the class atom-bond-configuration.

Attributes common to an atom-bond-configuration as well as methods operating on the instances of this class, are shown in Table II. Methods of particular interest include general setup methods, such as *map-old-abc-to-new-abc*, *equivalent-atom-bond-configuration-p*, and *identify-atom-bond-configuration-environment*. Setup methods, as expected, provide a means for instantiation. *Map-old-abc-to-new-abc* is a utility method that maintains the system pointers, which is an important feature when competing pathways are simultaneously analyzed. *Equivalent-atom-bond-configuration-p* determines when two instances of the atom-bond-configuration are equivalent, whereas *identify-atom-bond-configuration-environment* accesses information on the reaction environment. Although the reaction environment is specified for most pathways of synthetic interest, in the broader domain of computer-aided chemical reasoning many systems exist in which the environment is not known a priori.

2. *Modeling Elements Defining Reactive Behavior of Chemicals*

The reactive behavior of a chemical structure is determined by the interaction between low-level transformations of an instance of atom-

`bond-configuration (abc)`, such as bond cleavage, bond formation, electron distribution, and the conditions in the surrounding environment. To capture all the necessary information, LCR used the following classes of modeling elements.

a. Modeling Element 4: `chemical-behavior`. The electronic state that characterizes chemical reactivity is captured in chemical behavior. This may result from internal or external influences or both. Assessment of these electronic states characterize radical, nucleophilic, and electrophilic behavior or combinations thereof.

The set of behaviors that defines the spectrum of a species' reactivity is determined by its electronic state, which is dictated by internal and external influences. These influences may be established by the ground state, the excited state, or the effect of an external environment on the state. The chemical-behavior class uses a set of operations that define the chemical behavior of a chemical structure and proceed as follows:

Step 1. A set of operations, S, is used to deduce and assess the structural character, s, of an `abc` instance: $\texttt{abc} \xrightarrow{S} s$.

Step 2. Given s, a set of operations, I evaluates the electronic character i of the molecular structure: $s \xrightarrow{I} i$.

Step 3. Given a set of k species with structure $s_1, s_2, \ldots, s_k$, and electronic characters, $i_1, i_2, \ldots,$ a set of operations, E, assess the electronic character, e, of the surrounding environment,

$\{s_1, s_2, \ldots, s_k; i_1, i_2, \ldots, i_k\} \xrightarrow{E} e.$

Step 4. Given s, i, and e of a specific chemical structure, a set of operations B establishes the set of potential instances of `chemical-behavior` (or `cb`) for the specific chemical structure:

$\{s, i, e\} \xrightarrow{B} \texttt{cb}.$

The operations S, I, E, and B are methods invoked by the instances of the class `chemical-behavior` and form the basis for the assignment of specific reactivity of a given `abc`. During their execution call on other methods, they are encapsulated by the structural classes, `atom`, `bond`, and `abc`. An important feature of `cb` is that detail can be managed and persued on demand to assist in an evaluation. The structure permits the association of a set of potential behaviors with a species. These tasks can be performed at run time and the assignment made *dynamically*. For example, whether an alcohol acts as a bulk solvent, weak acid, weak nucleophile, or form an alcoholate anion, is dynamically assigned by `chemical-behavior` (cb) once the reaction environment is known.

b. Modeling Element 5: `reaction-environment`. The set of properties characterizing the environment in which a reaction is occurring is contained in reaction-environment. Attributes typical of this element include; "temperature," "pressure," "wavelength," "surface type," "species concentration," "pH," "species present," etc. Methods built into the modeling element evaluate attribute values and access information in other modeling elements.

c. Modeling Element 6: `ab-initio-operator`. Low-lying transformations that are characterized by bond cleavage, bond formation, and electron distribution are captured in the modeling element `ab-initio-operator`.These transformations may be grouped as mass transfer operations and energy transfer operations for abstraction purposes. Axioms, physical laws, rules, or constraints, obtained from the physical sciences are encoded to prevent the generation of an atom-bond-configuration obtained by applying infeasible transformations. Such knowledge may include conservation of mass, conservation of energy, charge support, Pauli exclusion principle, or valence constraints.

A specific chemical transformation or rearrangement is accomplished by calling on a set of base operations, called `ab-initio-operators` (K_{ai}'s), to perform the necessary elementary structural changes and electron redistribution. The base operations selected are evaluated on the reaction sites of the specified reactant(s). These low-lying operations have embedded models or constraints, such as conservation of mass and energy, that prevent the generation of infeasible structures as specified by the scope of knowledge that defines them. For example, if we have not specified operators capable of constructing delocalized bonds, then we should not expect structures that emanate from this knowledge. We have found that the knowledge embedded in these models is nearly always structural in nature reflecting for example, knowledge about orbital symmetry, atom coordination, or charge. As will be discussed later, K_{ai} operators can be used directly to generate the upper bound of theoretically feasible transformations.

d. Modeling Element 7: `composite-operator`. The class `composite-operator` contains knowledge about user specifications and mechanistic operations. Generic rate information, such as relative magnitude of rate constants (e.g., rate constants of photochemical reactions lie between

10^{10} and 10^{12} s^{-1}) and electronic state information (e.g., excited, radical, charged) are also contained in `composite-operator`. This information is used to limit an operator's range of applicability.

The class `composite-operator` contains procedures that transform a set of chemical species of predisposed behaviors into a set of products, provided that an optional set of prespecified conditions is satisfied. These conditions can be specified by the user or imposed by the system. They may encompass virtually any symbolically encodable concept: structural character, chemical behavior, spatial orientation, reaction conditions, free-energy requirements, enthalpy considerations, toxicity, etc. An instance of the `composite-operator` (e.g., K) calls on the following procedures to carry out the corresponding tasks:

$K_{get\text{-}sites}$, to identify the potential sites for reaction, by assessing the chemical-behavior of each `atom-bond-configuration`.

K_t, to transform a set of potential reaction sites, utilizing instances (e.g., $\mathtt{K}_{\mathtt{ai}}$) of `ab-initio-operator`. Successful applications of $\mathtt{K}_{\mathtt{ai}}$ operators generate new instances of `atom-bond-configuration`.

K_f, to return a Boolean value of "true," when the encoded set of prespecified conditions are achieved, and "false," otherwise.

LCR uses a set of predefined instances of K to represent a set of known transformations utilizing specific ab-initio-operators. New instances K of composite operators can be easily built by the user through the aggregation of different sets of ab-initio-operators, using the facilities of LCR.

3. *Modeling Elements for Reactions and Pathways*

The successful application of a composite operator on a chemical structure, or a set of structures, constitutes a reaction:

$$\mathtt{R}\colon \{\mathtt{abc}_1\} \xrightarrow{K} \{\mathtt{abc}_2\}.$$

a. Modeling Element 8: `reaction`. This modeling class captures the information that is specific to a given transformation. An instance of `reaction` contains information about the species determining which are the reactants and which are the products. It also contains a reference to the instance of the composite operator K, which is responsible for the transformation, and the instance of `reaction-environment`, which supplies the reaction conditions.

Instances of `reaction` constitute the building blocks for the construction of a reaction pathway P, which is modeled as an ordered list of `reactions:`

$$\mathtt{P} \hat{=} \{R_1, R_2, \ldots, R_n\}, \text{ where } R_1\colon \{\mathsf{abc}_{i-a}\} \xrightarrow{K_i} \{\mathsf{abc}_i\}.$$

The transitivity of semantic relations in LCR (see Section I.C) allows the system to automatically construct an instance of `reaction` as the abstract representation of a pathway. For example, if $\mathtt{R}_1$: $\mathtt{A} \rightarrow \mathtt{C}$ and $\mathtt{R}_2$: $\mathtt{B} \rightarrow \mathtt{C}$, then LCR constructs automatically $\mathtt{R}_3$: $\mathtt{A} \rightarrow \mathtt{C}$ and includes it in the library of reactions for future pathway contraction.

b. Modeling Element 9: `context`. A context is a consistent set of assumptions that characterize a species behavior or structural character. If one wants to change the description of some modeling elements (i.e., associate a new set of assumptions with them), but also preserve the former version, then one assigns a new `context` to the new version, keeping all other information the same. The use of context is particularly helpful when a reactant can exist in several forms, such as resonance structures, or keto enol forms. If a context is associated with a pathway, assumptions leading to the selection of a particular behavior can be modified and the pathway adjusted accordingly without reinitializing the entire system.

4. Modeling Elements to Describe Quantitative Relationships

In all the previous modeling elements, the need arises for establishing equations, inequalities, Boolean, ordinal or order-of-magnitude relationships among the attributes (variables and parameters) of the modeling elements. To capture these quantitative relationships, LCR uses the modeling classes `constraint` and `generic-variable` as well as their various subclasses. Since these modeling elements have been borrowed from the modeling language MODEL.LA., we will defer their description to Section IV. For the time being it suffices to say that `generic-variable` provides a structured representation of any variable or parameter, and `constraint` does the same for a set of different types of relationships.

5. Subclasses of Modeling Elements

Each of the nine modeling elements described above possesses a basic structure of attributes and methods and are inherited by any of their instances. Nevertheless, we have found that the construction of specialized subclasses, emanating from a specific class, enriches the vocabulary of

LCR and brings it closer to the linguistic constructs used by chemists and engineers. For example, a functional group can be described by an instance of the class `atom-bond-configuration`, but having a distinct modeling element, called `group`, with its own specialized attributes and methods, it facilitates the identification of such groups in molecules and streamlines the reasoning around such specific structures. Nevertheless, in order to preserve the fact that a group is also in atom-bond-configuration, we create the class `group`, as a subclass of the class `atom-bond-configuration`. Inheritance mechanisms of object-oriented programming allow the modeling class, `group`, to inherit all attributes and methods of the mother-class, `atom-bond-configuration`. Using this inheritance mechanism, we have developed expanded trees of modeling subclasses for four of the basic modeling elements: `atom-bond-configuration`, `chemical-behavior`, `ab-initio-operator`, and `composite-operator`.

a. The `atom-bond-configuration` *Classes.* Four main subclasses emanate from the `abc` class: (1) `group`, representing a substructure of the atom-bond-configuration—Additional groups may be added to or built from the list of conventional functional groups (e.g., acid from oxo and alcohol); (2) `ion`, representing any atom-bond-configuration that has a net electrical charge; (3) `radical`, representing any atom-bond-configuration that has an open shell; and (4) `molecule`, representing any atom-bond-configuration that is not an ion or a radical. Further specialization of these primary subclasses, either through the mixing of the primary subclasses (e.g., the class radical-ion may be formed by mixing radical and ion) or by specializing the class description (e.g., creating a class organic under molecule), allows extension of the generic abc-class.

Consider, for example, specialization of the class `molecule` (Fig. 3) into `organic-molecule` and `inorganic-molecule` and suppose `inorganic-molecule` contains a method for electron counting. This method is inherited to `organometallic-molecule`; how the method is used and combined with other methods is controlled by `organometallic-molecule`. Likewise, if methods and attributes of `organic-molecule` facilitate reasoning about the organometallic system, they can be included as well.

b. The `chemical behavior` *Classes.* Chemical behavior is classified into two main subclasses: `external-influences` and `internal-influences` (Fig. 4). Unlike the abc-class, the cb-class structure provides a means for communicating attribute values between classes, independent

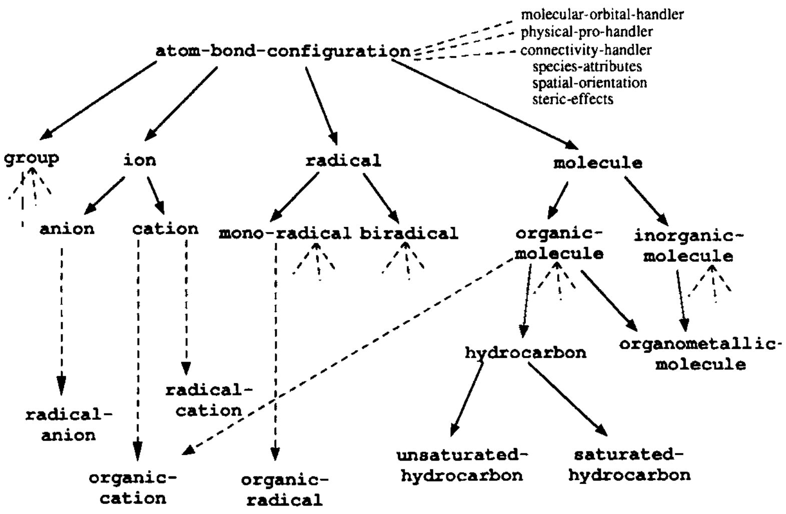


FIG. 3. Hierarchy of `atom-bond-configuration` subclasses.

of their position within the class structure. This is necessary because chemical behavior is often a combination of effects (e.g., the ground state of a species is influenced by its inherent electronic structure and its external environment). LCR provides a means of combining these effects. These operations constitute the basis for reactivity assignment. They are distributed throughout the hierarchy and are used to classify electronic states. The classes composing this hierarchy categorize electronic state and not the derived properties associated with an electronic state. Nucleophilicity, for example, is derived from the class `pertinent-occupied-molecular-orbital` (POMO). Radical behavior is derived from the class `singly-occupied-molecular-orbital` (SOMO).

As in the abc hierarchy, each class composing the chemical behavior hierarchy contains only those methods and attributes that pertain to it. Concepts derived at a particular class are used in classes of higher specialization so that more sophisticated concepts can be deduced and discriminating properties elucidated at the proper level. For example, the class POMO contains methods for evaluating the POMO, whether it is the HOMO, the n-HOMO, or any other occupied molecular orbital. These attributes may then be used at more specialized level to expand on the features of an electronic state that gives rise to a particular conceptual behavior. Nucleophilicity illustrates this point, because it may arise from either a negatively charged ion or a neutral atom (i.e., nucleophilicity can

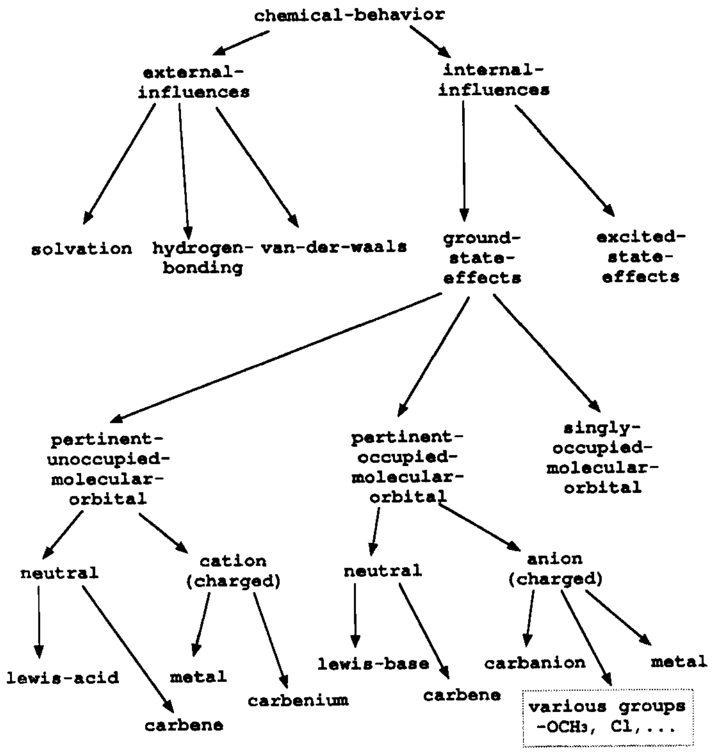


FIG. 4. Hierarchy of chemical-behavior subclasses.

result from lone pairs that are bonded, as in the case of the lone pairs on nitrogen; or nonbonded, as in the case of the electrons comprising a π bond).

An important distinction between the abc and cb hierarchy is that the cb hierarchy of classes is not treated as task specialization by operators calling for the assignment of chemical behavior. Since cb is a set of potential behaviors, {b}, it is necessary to associate behaviors to a species that lies on the same class level [e.g., pertinent unoccupied molecular orbital (PUMO) and POMO] as well as on different, more specialized, levels of the same class (e.g., Lewis-base and POMO). This is necessary because the species dynamically assumes the requisite behavior—(weak) nucleophile or Lewis base—when it is required by the procedure $K_{get\text{-}sites}$.

c. The ab-initio-operator *Classes.* Five primary subclasses were developed to model elementary transformations (Fig. 5) (1) bond-

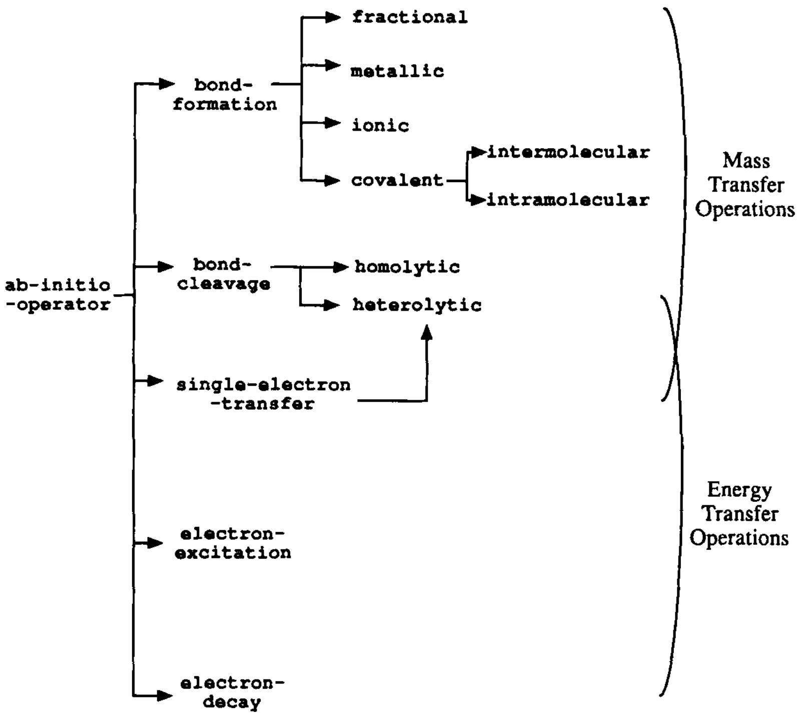


FIG. 5. Hierarchy of ab-initio-operator subclasses.

formation, (2) bond-cleavage, (3) single-electron-transfer, (4) electron-excitation, and (5) electron-decay. The operations stem from two types of operations: mass transfer and energy transfer. Each subclass contains methods that prevent the generation of theoretically infeasible structures. As in the abc-class, the subclasses of ab-initio-operator can be mixed, together with their methods, to form various new classes. For example, the subclass heterolytic-bond-cleavage is formed by combining the parent classes of single-electron-transfer and bond-cleavage. Together, these operations afford the generation of any elementary reaction mechanism.

d. The composite-operator *Classes.* The hierarchy of composite-operator classes is shown in Fig. 6. As described earlier, composite operators are built from sets of ab-initio-operators. They utilize spatial

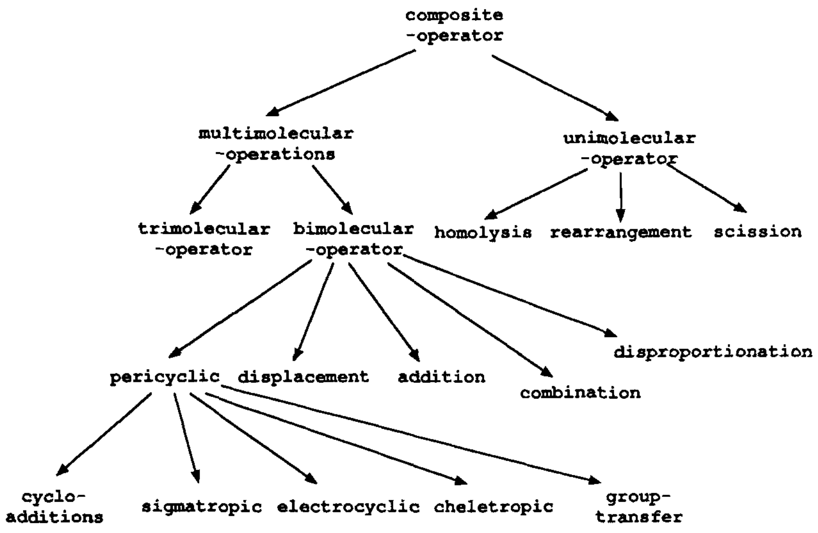


FIG. 6. Hierarchy of `composite-operator` subclasses.

operators, which modify the orientation of an `abc`. Although many hierarchies can be developed utilizing various classification schemes, for efficiency reasons, we have chosen a two-staged classification built on particle reaction requirements and specialized by structural reaction requirements. Because of this formulation, there are only two primary subclasses extending from `composite-operator`: `multimolecular-operator` and `unimolecular-operator`. Each subclass is then further specialized according to the structural requirements of the composite operator. For example, the bimolecular operation of radical coupling requires two radical substrates.

This classification limits the combinatorial explosion encountered in conventional computer-aided synthesis approaches, because composite operators know a priori when they apply. For example, the application of bimolecular operators to a list of chemical species results in the selective application of `radical-operator` to radicals contained in the list and `molecule-operator` to species that are molecules.

B. Semantic Relations among Modeling Elements in LCR

If the modeling elements of LCR represent the modes of a network, the semantic relationships constitute the links (edges) of the network. There exist different types of links, with each one establishing a different interpretation on the relationship of the connected nodes. Thus, whereas the modeling objects carry by themselves no assertional importance, the semantic relationships establish assertions and thus impose interpretation, i.e., meaning, to the representational objects.

1. Entity-Attribute Semantics

The first two semantic relationships establish the structure of a modeling element; they allow the declaration of the attributes and methods of a modeling element. Although all object-oriented systems possess, by definition, the first two semantic relationships and no special computer-aided provisions are needed, they have been included here for completeness.

*a. Semantic Relationship 1: "**is-attribute-of.**"* This relationship enables the association of a modeling object as an attribute of an other object. It is useful when the present scope of LCR's modeling elements needs to be extended. Without this relationship a modeling would have remained a conventional low-level data structure. For example, the attributes of the object representing hydrocarbons are declared as follows:

`hydrogen-atoms`	***is-attribute-of***	`hydrocarbon`
`aromatic-p`	***is-attribute-of***	`hydrocarbon`

*b. Semantic Relationship 2: "**is-method-of.**"* This relationship declares that a given computational procedure is "owned" by a particular modeling object, and that the procedure describes part of the object's behavior or computational ability. The following are typical examples of the use of this semantic relationship:

compute-aliphatic-p	***is-method-of***	`hydrocarbon`
collect-longest-carbon-chain	***is method-of***	`hydrocarbon`

2. Specialization Semantics

The following two semantic relationships establish the links between a basic modeling element and its derivative subclasses of modeling objects and instances. Both of them are isomorphic mappings.

*a. Semantic Relationship 3: "**is-a.**"* This is needed to establish *sub/superset* links and is used to define various new subclasses of modeling objects, emanating from a specific class. Thus, starting with a small number of basic modeling elements, this semantic relationship allows a modeling language to be expanded and remain open-ended. Each new class of modeling objects is a structure isomorphic to the modeling element from which it was derived, but with more specializations. Typical examples of its use are

`molecule`	*is-a*	`abc`
`hydrocarbon`	*is-a*	`organic-molecule`
`addition`	*is-a*	`bimolecular operator`

*b. Semantic Relationship 4: "**is-a-member-of**."* This relationship is needed to relate isomorphic structures, emanating from the same class of modeling objects, with identical specialization. For example, assume that specific assumptions on the modeling of free radicals lead to the class of modeling objects called `radical`. Then if specific free radicals, `radical-1` and `radical-2`, are modeled under the same set of assumptions, the resulting models will have the same internal structure as `radical`, but different assigned names and values to the various variables or attributes describing it. Consequently

`radical-1`	***is-a-member-of***	`radical`
`radical-2`	***is-a-member-of***	`radical`

3. Specification Semantics

Specification mappings, described by the following six semantic relationships, are used to specify the value of attributes of various modeling elements. They express, (a) binary relations, such as whole/ part links, (b) communication lines among modeling objects, or (c) the value of simple describing properties.

*a. Semantic Relationship 5: "**is-composed-of**."* It defines the link between a modeling object and other modeling objects in which the former is decomposed. For example, to represent a `free-radical-reaction`, one may want to disaggregate it to the level of `initiation-reaction`, `propagation-reaction`, and `termination-reaction`. In such case

free-radical-reaction	***is composed-of***	{initiation-reaction; propagation-reaction; termination reaction}

*b. Semantic Relationship 6: "**is-part-of.**"* This relationship defines the link between a modeling object and the modeling object that is containing it. Continuing with the example presented above, we can specify that

initiation-reaction ***is-part-of*** free-radical-reaction

In particular, this semantic relation is established when the attribute "parent" of the modeling element `initiation-reaction` receives the value `free-radical-pathway`. The "***is-part-of***" relationship deals with structural containment—the "***is-a***" relationship describes conceptual containment. It is obvious that one of the properties of the relation is *transitivity*, and that "***is-composed-of***" and "***is-part-of***" are symmetrical relationships.

*c. Semantic Relationship 7: "**is-attached-to.**"* This is the first of two semantic relationships used to elucidate the structural connectivity among specific modeling elements, such as *atoms* to form *molecules*. A typical example is

`oxygen-1` ***is-attached-to*** `(carbon-1; carbon-2)`

The ***is-attached-to*** semantic relationship establishes linkages between different modeling elements, allowing flow of information from one to the other(s). Furthermore, since atoms know their membership into various functional groups, auxiliary links are automatically established; thus LCR creates automatically the following semantic connection:

`oxygen-1` ***is-attached-to*** `oxo-1`

or

`oxygen-1` ***is-attached-to*** `carboxyl-1`

*d. Semantic Relationship 8: "**is-connected-by.**"* This is the symmetric relationship of ***is-attached-to***. Using the previous examples we have

`oxygen-1` ***is-connected-by*** `carbon-1`
`oxygen-1` ***is-connected-by*** `carbon-2`

*c. Semantic Relationship 9: "**is-described-by.**"* At a basic level, the state of a modeling object is described by the values of a set of variables. Moreover, its function or behavior is described by mathematical relationships (or procedures). This semantic relationship is used to declare the variables and mathematical relationships which describe the state of a modeling object, for example:

`oxygen-1`	***is-described-by***	"hybridization-oxygen-1"
$\mathtt{S_n2}$	***is-described-by***	*K-get-sites-*S_n*2.*

The first example indicates that the object oxygen-1 is described by the value of the attribute "hybridization", and the second example shows that the behavior of object, S_n2 (an instance of class `reaction`) is described by the procedure *K-get-sites-*S_n*2.*

*f. Semantic Relationship 10: "**is-describing.**"* This is the inverse of the previous relationship:

"hybridization-oxygen-1"	***is-describing***	`oxygen-1`
K-get-sites-S_n*2*	***is-describing***	$\mathtt{S_n2}$

LCR has the appropriate mechanisms to keep symmetrical semantic links properly updated, when one of them changes (e.g., is created, deleted, or modified).

*g. Semantic Relationship 11: "**is-characterized-as.**"* This semantic relationship specializes a class by defining the character (i.e., value) of some of its properties. It represents the semantic relation between a modeling object and an attribute of its description. For example, if the class `pertinent-occupied-molecular-orbital` is described as homolytic, and nucleophilic, semantic links like

POMO	***is characterized-as***	`homolytic`
POMO	***is-characterized-as***	`nucleophilic`

are established between the modeling objects and the attributes of its description. This semantic relationship is particularly important in defining the context of a particular pathway. For example, after a pathway is constructed, the user may wish to change the characterization of an intermediate (e.g., Lewis acid versus nucleophile) so that the implication of the change can be investigated. This is accomplished easily with the semantic relationship ***is-characterized-as***. The axioms of commutativity and merging are supported by this relation.

4. *Aggregation / Disaggregation Semantics*

Aggregation (or, abstraction) is the process by which details of a modeling object are left unspecified in favor of a less cluttered description of its structure. Disaggregation (or refinement) is the opposite process. Thus, abstraction maps a set of modeling objects into an object described by a simpler modeling element.

*a. Semantic Relationship 12: "**is-disaggregated-in.**"* This semantic relation exists between modeling elements located in different contexts and responds to the need of breaking down systems into smaller, more tractable components, where additional detail can be added. For example, a theoretically feasible but unsubstantiated pathway can be characterized by a set of assumptions that lead to the structural character defining the constituent intermediates. This same pathway may be characterized by another set of assumptions, at a later stage, that leads to a different set of intermediates. Different contexts allow us to encapsulate the distinct representation. However, if we seek to transfer information from the pathway to the species in which it is broken we have to create a communication route. The semantics links

global-pathway-1	***is-disaggregated-in***	initiation-pathway-1
global-pathway-1	***is-disaggregated-in***	propagation-pathway-1
global-pathway-1	***is-disaggregated-in***	termination-pathway-1

provide the communication vehicle.

Notice that the ***is-disaggregated-in*** link is designed to support communications between objects of the *same type in different contexts*. In this way it is differentiated from ***is-composed-of***. The scope of ***is-composed-of*** is restricted to objects of any type provided they are in the *same context*. The behavior of ***is-disaggregated-in*** is similar to that of ***is-composed-of*** with respect to the properties that it supports. This semantic link obeys the axioms of transitivity, commutativity, and merging.

*b. Semantic Relationship 13: "**is-abstracted-by.**"* It is the inverse of the previous one, and using the same example, we have

{initiation-pathway-1, propagation-pathway-1, termination-pathway-1}

is-abstracted-by global-pathway-1

5. Properties of LCR's Semantic Relationships

The semantic relations of LCR establish how different objects relate to one another. They were formally defined to obey the following three axioms:

1. *Transitivity*:

    If (O1 "semantic-relation" O2) and (O2 "semantic-relation" O3)
    then
    (O1 "semantic-relation" O3)

   where "semantic-relation" applies to the relations `is-a`, `is-composed-of`, and `is-part-of`, and O1, O2, and O3 are arbitrary modeling objects of this system.

   For example,

If (`Methyl-1` *is-part-of* `Ethyl-1`) and (`Ethyl-1` *is-part-of* `Ethane-1`)
then
(`Methyl-1` *is-part-of* `Ethane-1`)

2. *Commutativity*:

(THE "*attribute-1*" OF O1 IS A_1)
. . . .
(THE "*attribute-1*" of O1 IS A_{j-1})
(THE "*attribute-1*" of O1 IS A_j)
. . . .
(THE "*attribute-1*" of O1 IS A_n)

is the same as

(THE "*attribute-1*" of O1 IS A_1)
. . . .
(THE "*attribute-1*" of O1 IS A_j)
(THE "*attribute-1*" of O1 IS A_{j-1})
. . . .
(THE "*attribute-1*" of O1 IS A_n)

where "*attribute-1*" is any specific attribute establishing one of the following semantic relationships: "***is-composed-of***," "***is-attached-to***," "***is-connected-by***," "***is-described-by***," "***is-describing***," and "***is-characterized-as***." In other words, the order in which attributes are listed is irrelevant.

3. *Merging*:

(THE "*attribute-1*" OF O1 IS A_1)
. . . .
(THE "*attribute-1*" of O1 IS A_{j-1})
(THE "*attribute-1*" of O1 IS A_j)
. . . .
(THE "*attribute-1*" of O1 IS A_n)

is the same as

(THE "*attribute-1*" of O1 IS SET.OF ($A_1 \ldots A_{j-1}\ A_j \ldots A_n$))

where "*attribute-1*" is any specific attribute establishing one of the following semantic relationships: "***is-composed-of***," "***is-attached-to***," "***is-connected-by***," "***is-described-by***," "***is-describing***," or "***is-characterized-as***." Hence, attributes of the same concept can be merged. For example,

(THE Neighbor-atoms of `Carbon-1` IS `Hydrogen-17`)
(THE Neighbor-atoms of `Carbon-1` IS `Nitrogen-21`)
(THE Neighbor-atoms of `Carbon-2` IS `Carbon-2`)

is the same as

(THE Neighbor-atoms of `Carbon-1`
IS THE SET OF
(`Hydrogen-1, Nitrogen-21, Carbon-2`))

C. Syntax of LCR

Grammars are intended to capture what we may call the *structure* or *syntax* of languages, as opposed to their meaning or semantics. The syntax of a programming language is a strict, precise set of rules that describe the string of symbols that constitute legal statements and specify how a statement breaks down into its constituent parts. The syntax description is done using formal grammar, often referred as *metalanguage* (i.e., special language used to describe other languages). The description language used here is an extended BNF (Backus–Naur form or Backus–normal form), which is more compact than BNF. The BNF (Naur, 1963) is a widely used formal method developed by computer scientists for the precise syntactic description of computer languages. The BNF grammars conventionally have four elements: (1) a *terminal vocabulary* that corresponds directly to the vocabulary of allowed tokens of the language to be defined; (2) a *nonterminal vocabulary*, conventionally enclosed in pointed brackets (⟨ and ⟩); (3) a *set of production rules* defining way of building up phrases (represented by nonterminal symbols) from terminal and nonterminal vocabulary items; and (4) a *correspondence* indicating which nonterminal symbol is associated with the "master" or "sentence" phrase type of the language. In defining the grammar we will follow the standard practice of writing a production beginning with the "master" phrase type at the top of the grammar.

Two important observations can be made about BNF grammar rules. First, BNF rules can be defined in a recursive manner; that is, the phrase name can appear on both sides of the symbol "::= ." This property can be observed in the rules defining "⟨structural-def⟩," "⟨input-def⟩," "⟨output-def⟩," etc. This recursive property allows an infinite number of statements to be described by a finite number of rules. Second, a BNF description of a grammar is complete when every phrase name has been defined reaching the level of terminal vocabulary.

1. Explanation of Notation

::=	Is defined as.
[ ]	Encloses optional unit.
{ }	Indicates mandatory choice.
⟨ ⟩	Unit that is described separately.
...	Indicates repetition of syntactic signs.
|	Separator for alternatives.
*	What the braces enclose may appear any number of times (including zero).
+	What the braces enclose may appear any non-zero number of times (must appear at least once).
[[ ]]	Double brackets indicate that any number of the alternatives enclosed may be used, and those may occur in any order, but each alternative may be used at most once unless followed by a star.

2. Examples of Syntax

Word in capital letters: reserved word.

Lowercase words: metalinguistic variable names.

```
⟨modeling-element⟩::= ⟨modeling-element-name⟩[⟨attribute⟩]*.
⟨modeling-element-name⟩::= (⟨modeling-class⟩|⟨atom⟩|
                            ⟨bond⟩|⟨reaction⟩|
                            ⟨reaction-environment⟩|⟨context⟩)
                           [attribute⟩]*[⟨method⟩]+.
⟨attribute⟩::= ⟨attribute-name⟩ IS-ATTRIBUTE-OF
               ⟨modeling-element⟩.
⟨method⟩::= ⟨method-name⟩ IS-METHOD-OF ⟨modeling-element⟩.
⟨class⟩::= (⟨class-name⟩ IS-A {[⟨modeling-class⟩]+||[⟨class⟩]+}).
⟨instance⟩:: + (⟨instance-name⟩ IS-MEMBER-OF {[⟨class⟩]+}.
```

⟨abc⟩::= ⟨abc-input⟩.
⟨abc-input⟩::= be-matrix|atom-table bond-table.
⟨abc⟩::= ⟨group⟩|⟨ion⟩|⟨radical⟩|⟨molecule⟩.
⟨group⟩::= [atom]$^+$[bond]$^+$.
⟨ion⟩::= [atom]$^+$[bond].
⟨radical⟩::= [atom]$^+$[bond].
⟨molecule⟩::= [atom]$^+$[bond]$^+$.
⟨cb⟩::= {[⟨behavior⟩]$^+$}.
⟨behavior⟩::= (IS-CHARACTERIZED-BY {⟨Internal-Influences⟩}
[⟨External-Influences⟩]).
⟨Internal-Influences⟩::= (IS-CHARACTERIZED-BY
{⟨Ground-state-effects⟩
⟨Excited-state-effects⟩}).
⟨Ground-state-effects⟩::= (IS-CHARACTERIZED-BY
{⟨PUMO⟩|⟨POMO⟩|⟨SOMO⟩}).
⟨Excited-state-effects⟩::= (IS-CHARACTERIZED-BY {$\sigma^*|\pi^*$}).
⟨External--Influences⟩::= (IS-CHARACTERIZED-BY
{[⟨solvation⟩][⟨hydrogen-bonding⟩]
[⟨Van der Waals⟩]}.
⟨composite-operator⟩::= ⟨inputs⟩⟨enabling-conditions⟩
⟨abinitio-operator⟩⟨output⟩.
[⟨composite-operator⟩].
⟨input⟩::= ⟨react-cond⟩⟨abc⟩⟨cb⟩.
⟨output⟩::= ⟨abc⟩.
⟨abc⟩::= {[(Instance-of-abc)]$^+$}.
⟨cb⟩::= {[(Instance-of cb)]$^+$}.
⟨reaction-cond⟩::= {(Instance-of reaction-conditions)}.
⟨enabling-conditions⟩::= {[⟨conditionals⟩]$^+$}.
⟨conditionals⟩::= {[[(Instance-of user preferences)]$^+$
{[(Instance-of physical requirements)]$^+$}]}.
⟨abinitio-operator⟩::= {[⟨Kai⟩]$^+$}.
⟨K_{ai}-operator⟩::= Bond formation|Bond Cleavage|
Single Electron Transfer|Electron Excitation|
Electron Decay.
⟨K_{ai}⟩::= ⟨K_{ai}-operator⟩⟨ – ⟨K_{ai}-input⟩⟨K_{ai}-enabling-cond⟩⟨output⟩.
⟨K_{ai}-input⟩::= {[⟨reaction-center⟩]$^+$}.
⟨reaction-center⟩::= {[⟨atom⟩]$^+$}.
⟨K_{ai}-enabling-cond⟩::= {[⟨K_{ai}-condition⟩]$^+$}.
⟨K_{ai}-condition⟩::= {[physico-chemical-requirements]$^+$}.
⟨physical-requirements⟩::= (IS-ABSTRACTING
⟨physio-chemical-requirements⟩).

III. Formal Construction of Representations for Chemicals and Reactions

The nine basic modeling elements of LCR, although they offer a fairly limited vocabulary, are generic enough to allow (1) *infinite extensibility* of LCR's vocabulary through a finite set of rules, and (2) *representation and analysis* of any potential reaction pathway.

In this section we will examine the mechanisms that LCR possesses to achieve these objectives.

A. Extension of LCR's Modeling Objects

In Section II.B we discussed the modeling subclasses emanating from the four basic elements: `atom-bond-configuration`, `chemical-behavior`, `ab-initio-operator`, and `composite-operator`. Let us examine what mechanisms LCR possesses for such linguistic extensibility, which is imperative for the representation of the widely diverse knowledge of chemistry.

1. *Extension of the* `atom-bond-configuration` *Hierarchical Tree*

Suppose it is advantageous to reason about cyclic aliphatic species directly. For this purpose, we create the modeling class `cyclic-aliphatic` and link it to the parent class `aliphatic` (Fig. 7), thereby establishing a set membership:

cyclic-aliphatic ***is-a*** aliphatic

Since `aliphatic` is a specialization of the class `hydrocarbon`, it is advantageous to create a link between `aliphatic` and `hydrocarbon` (Fig. 7):

aliphatic ***is-a*** hydrocarbon

This allows the attributes describing hydrocarbon to be inherited by aliphatic. Similarly, the link established between `aliphatic` and `cyclic-aliphatic` allows this information to be accessed from `cyclic-aliphatic` directly. Having created these links, any attribute or method residing in the class `aliphatic` or any of its superclasses is inherited by `cyclic-aliphatic`. For example, if methods *cyclic-p*,

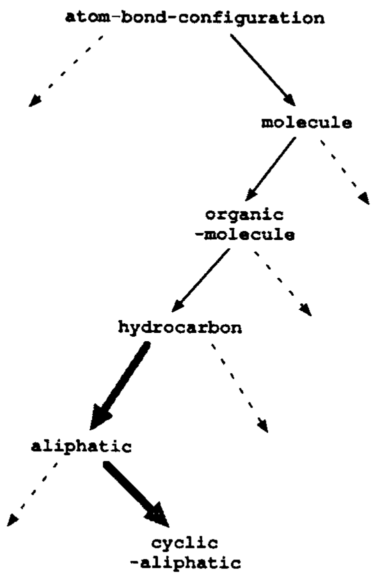


FIG. 7. Extending the `atom-bond-configuration` tree of subclasses.

compute-nonbonded-strain, and *compute-torsional-strain* reside in `atom-bond-configuration` and methods *compute-skeleton-chains* and *compute-longest-carbon-chain* reside in `organic-molecule`, each of these methods as well as any additional descriptive attributes are directly accessible from the modeling class `cyclic-aliphatic`; no duplication of modeling effort is required. How methods and attributes are combined or mixed into the `cyclic-aliphatic` class is user-controlled, through LCR's semantic relationships.

It may also be desirable to extend the utility of the `cyclic-aliphatic` class and its descriptors through the addition of class-specific methods and attributes (Table III). For example, the user may wish to include the following attributes in the class description:

ring-strain	***is-attribute-of***	`cyclic-aliphatic`
number-of-rings	***is-attribute-of***	`cyclic-aliphatic`
ring-sizes	***is-attribute-of***	`cyclic-aliphatic`
ring-number	***is-attribute-of***	`cyclic-aliphatic`
torsional-strain	***is-attribute-of***	`cyclic-aliphatic`
nonbonded-strain	***is-attribute-of***	`cyclic-aliphatic`
...		

TABLE III

SELECT ATTRIBUTES AND METHODS FOR `cyclic-aliphatic` AND `excited-state-effects` CLASSES

`cyclic-aliphatic`	`excited-state-effects`
cyclic-aliphatic attributes	**excited-state-effects attributes**
identifier (inherited from super class)	identifier (inherited from super class)
old-identifier (inherited from super class)	old-identifier
	(inherited from super class)
name (inherited from super class)	bonding-orbital
	(inherited from super class)
parent-abc (inherited from super class)	anti-bonding-orbital
	(inherited from super class)
progenitors (inherited from super class)	dominate-behavior
	(inherited from super class)
...(inherited from super class)	competitive-behavior
	(inherited from super class)
ring-strain	...(inherited from super class)
number-of-rings	S_0
ring-sizes	S_1
ring-number	S_2
torsional-strain	S_3
nonbonded-strain	T_0
bridged-atoms	T_1
...	T_2
cyclic-alphatic methods	...
compute-identifier (inherited from super class)	**excited-state-effects methods**
compute-old-identifier (inherited from super class)	compute-identifier
	(inherited from super class)
compute-parent-abc (inherited from super class)	compute-old-identifier
	(inherited from super class)
compute-progenitors (inherited from super class)	compute-bonding-orbital
	(inherited from super class)
...(inherited from super class)	compute-anti-bonding-orbital
	(inherited from super class)
compute-name (super class method modified)	compute-dominate-behavior
compute-ring-strain	(inherited from super class)
compute-number-of rings	compute-competitive-behavior
compute-ring-sizes	(inherited from super class)
compute-ring-number	...(inherited from super class)
compute-torsional-strain	compute-S_0
compute-nonbonded-strain	compute-S_1
compute-bridged-atoms	compute-S_3
monocyclic-ring-p	compute-T_0
bicyclic-ring-p	compute-T_1
fused-ring-p	compute-T_2
...	compute-T_3
	assess-energy-gap
	allowed-transition-p
	forbidden-transition-p
	fluorescence-p
	phosphorescence-p
	internal-conversion-p
	internal-system-crossing-p
	...

The values of these attributes are then established by the methods that have been associated with cyclic-aliphatic. Methods indicative of this class include the following:

compute-ring-strain	***is-method-of***	cyclic-aliphatic
compute-number-of-rings	***is-method-of***	cyclic-aliphatic
compute-ring-sizes	***is-method-of***	cyclic-aliphatic
compute-nonbonded-strain	***is-method-of***	cyclic-aliphatic
compute-bridged-atoms	***is-method-of***	cyclic-aliphatic
monocyclic-ring-p	***is-method-of***	cyclic-aliphatic
bicyclic-ring-p	***is-method-of***	cyclic-aliphatic
fused-ring-p	***is-method-of***	cyclic-aliphatic

In these methods, those with the "compute-" prefix determine attribute values; those with the "-p" suffix are predicate methods returning Boolean values. The remaining methods are invoked on instances of the class cyclic-aliphatic and are designed to raise the abstraction level (i.e., they allow the user to communicate using higher-level concepts). These methods may be called from outside of the class cyclic-aliphatic provided they operate on instances of the cyclic-aliphatic class. The method *bridged-atoms* illustrates this concept; it calls *bridged-atom-p*, a predicate method of atom, from the class cyclic-aliphatic. Likewise, if reactivity assessment requires knowledge about ring structure, we can access that information by applying the semantic relation ***is-describing*** to an instance of cyclic-aliphatic (ICA), as shown below:

bridged-atoms-ICA ***is-describing*** (atom-1...atom-n)

Notice also that although the methods, *compute-nonbonded-strain* and *compute-torsional-strain* were inherited by cyclic-aliphatic, more sophisticated versions of these methods were supplied to cyclic-aliphatic. This would have been unnecessary if the original versions of these methods supported fully *all* cyclic configurations (i.e., a robust method given at any ring configuration). We have found that the ability to incrementally improve methods accelerates model development.

2. *Extension of the* chemical behavior *Hierarchical Tree*

Suppose that we choose to embellish the modeling class chemical-behavior by developing the subclass excited-state-effects. We begin by developing a subclass architecture consistent with the existing organization: characterization of the electronic state. In this spirit, we have specialized the class internal-influences into the modeling

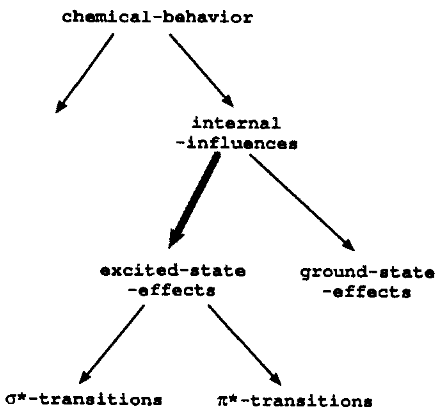


FIG. 8. Extending the `chemical-behavior` tree of subclasses.

subclasses `ground-state-effects` and `excited-state-effects`. We do this using the semantic relation ***is-a*** (Fig. 8):

excited-state-effects	***is-a***	internal-influence

Further, we specialize `excited-state-effects` according to the accompanying electronic transitions: transitions to σ^* and transition to π^*; and link these subclasses to excited-state-effects as shown below (see also Fig. 8):

σ^*`-transitions`	***is-a***	`excited-state-effects`
π^*`-transitions`	***is-a***	`excited-state-effects`

The addition of descriptive attributes and methods to each of these new modeling classes further characterizes each class. To characterize the modeling class `excited-state-effects`, we add attributes and methods descriptive of singled (S) and triplet (T) states. The states are refined further by energy level. For example ground states (S_0, T_0) are differentiated from their excited states (S_1, T_1) as well as from their higher states (S_2, T_2, and S_3, T_3). As before, we link these attributes using the semantic relationship ***is-attribute-of***:

S_0 (or, S_1, S_2, S_3)	***is-attribute-of***	`excited-state-effects`
T_0 (or, T_1, T_2, T_3)	***is-attribute-of***	`excited-state-effects`

Additional attributes may also be added. The degree to which it is advantageous to add additional class descriptors (e.g., attributes, methods) is defined by the scope of the modeling efforts.

Methods characterizing `excited-state-effects` are linked to their associated class using the semantic relation ***is-method-of***. These methods can be grouped according to whether the result is used by the attribute's compute method:

compute-S_0 (or, -S_1, -S_2, -S_3)	***is-method-of***	`excited-state-effects`
compute-T_0 (or, -T_1, -T_2, -T_3)	***is-method-of***	`excited-state-effects`

or by methods used to derive additional properties of the attribute space. The latter group may include generic methods such as *assess-energy-gap*, which computes the energy difference between two energy states (e.g., S_1 and S_2 or S_1 and T_1); or predicate functions that access feasibility. Several of these are shown below:

assess-energy-gap	***is-method-of***	`excited-state-effects`
allowed-transition-p	***is-method-of***	`excited-state-effects`
forbidden-transition-p	***is-method-of***	`excited-state-effects`
fluorescence-p	***is-method-of***	`excited-state-effects`
internal-conversion-p	***is-method-of***	`excited-state-effects`
internal-system-crossing-p	***is-method-of***	`excited-state-effects`

Together these elements give the modeling class' utility. Notice that the capability of moving electrons between energy levels requires that `atom` or `bond` have attributes that describe atomic and molecular orbitals. Given these attributes, the movement of an electron from one energy level to another can be orchestrated by methods residing in `atom`, `bond`, and `ab-initio-operator` (e.g., *ionization* or *single-electron-transfer*). In the latter case, a well-defined abstraction barrier is established to help maintain program modularity. Clearly, if the modeling effort does not require analysis of electron movement, the functionality need not be included. If it is required, LCR has the capability to expand and accommodate the modeling effort regardless of scope. A state description of `excited-state-effects`, after attribute and method addition, is shown in Table III.

Similarly, the subclasses of `excited-state-effects`, i.e., σ^*-transitions and π^*-transitions, are described by the transitions which characterize them. These include σ to σ^*, n to σ^*, n to π^*, and π to π^* transitions. The corresponding attributes and methods are linked to their respective classes as described earlier. Very briefly, they are shown below:

$\sigma \rightarrow \sigma^*$	***is-attribute-of***	σ^*`-transitions`
$n \rightarrow \sigma^*$	***is-attribute-of***	σ^*`-transitions`
...		
$\pi \rightarrow \pi^*$	***is-attribute-of***	π^*`-transitions`
$n \rightarrow \pi^*$	***is-attribute-of***	π^*`-transitions`
...		

and

compute-$\sigma \rightarrow \sigma^*$-transition	***is-method-of***	σ^*`-transitions`
*compute-*n $\rightarrow \sigma^*$*-transition*	***is-method-of***	σ^*`-transitions`
...		
compute-$\pi \rightarrow \pi^*$*-transition*	***is-method-of***	π^*`-transitions`
*compute-*n $\rightarrow \pi^*$*-transition*	***is-method-of***	π^*`-transitions`
...		

We can also specialize a modeling class by characterizing several of its dominant or critical properties. For selected operations, such as composite or ab initio operations, specialization can further refine the search space and afford greater (search) efficiency.

The semantic relationship ***is-characterized-as*** is used for this purpose. It provides a window for viewing the more interesting behaviors/ structures associated with a class, such as

excited-state-transition	***is-characterized-as***	$\pi \rightarrow \pi^*$
excited-state-transition	***is-characterized-as***	n $\rightarrow \pi^*$

or

electronic-state	***is-characterized-as***	excimer
electronic-state	***is-characterized-as***	exciplex

Using this semantic relation, the user may investigate alternative pathways by specifying a different characterization. Consistency is maintained by the multiple-context modeling utility of LCR.

3. *Extension of the* `ab-initio-operator` *Hierarchical Tree*

Our attention now turns to the operators: methods that perform the actual chemical transformations. Bond formation, bond cleavage, electron excitation, and electron decay are a few such examples. These operators lie at the heart of LCR.

We will illustrate the development of these operators by creating the ab initio modeling element, `covalent-bond`. We will establish in this development: (1) the communication protocol between `covalent-bond` and `ab-initio-operator`, (K_{ai}); (2) the criteria on enabling conditions; (3) how criteria can be relaxed, tightened, augmented, or supple-

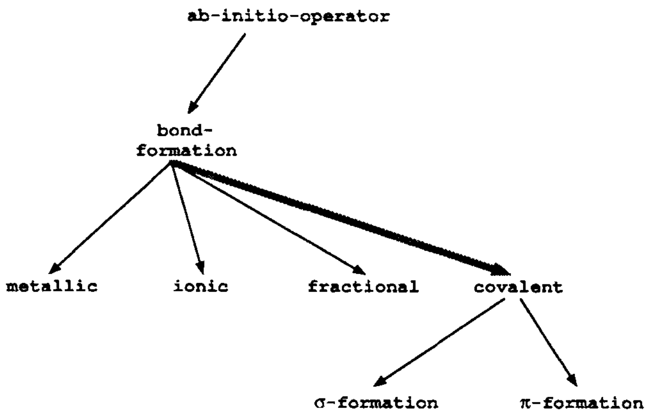


FIG. 9. Extending the ab-initio-operator tree of subclasses.

mented; and (4) how additional functionality can be added to covalent-bond (reflecting advances in our understanding of the bonding mechanism).

We initiate the development of covalent-bond by creating the modeling element and establishing the normal superclass links using the semantic relationship, ***is-a*** (Fig. 9):

covalent-bond	***is-a***	ab-initio-operator
σ-formation	***is-a***	covalent-bond
π-formation	***is-a***	covalent-bond

However, we will not specify the attributes of covalent-bond, σ-formation, or π-formation as we did previously, where the attributes were used for purposes of bookkeeping. How the attributes are chosen and ultimately used depends on the character and generality of the supporting encoded methods.

To effect a transformation, covalent-bond requires the use of specific methods. In particular, we need to establish generic methods that are responsible for primary transformations. The dominant method of covalent-bond is *make-covalent-bond*. *Make-covalent-bond* establishes a covalent-bond between two reaction centers when enabling conditions are achieved. It also performs all the necessary bookkeeping to ensure the bond is recognizable by the instance of abc. Expressed in pseudocode convention, these methods take the following form:

```
METHOD: make-covalent-bond
                    input
                    initialize
                    if   feasible-p
                         then enable-transformation
                         return
                    end
                    return
```

where the method *enable-transformation* performs the transformation and the method, *feasible-p*, represents the enabling conditions, $\langle K_{ai}$-enabling-cond$\rangle$, of the transformation.

The syntax for the definition of the subclass was given in Section II.B and is reproduced by the following seven (7) lines in Backus–Naur form (BNF):

1. $\langle K_{ai}\rangle$::= $\langle K_{ai}$-operator$\rangle\langle K_{ai}$-input$\rangle\langle K_{ai}$-enabling-cond$\rangle\langle$output$\rangle$
2. $\langle K_{ai}$-operator$\rangle$::= Bond formation | Bond Cleavage | Ionization | Single Electron Transfer | Electron Excitation | Electron Decay
3. $\langle K_{ai}$-input$\rangle$::= {[$\langle$reaction-center$\rangle$]$^+$}
4. $\langle$reaction-center$\rangle$::= {[$\langle$atom$\rangle$]$^+$}
5. $\langle K_{ai}$-enabling-cond$\rangle$::= {[$\langle K_{ai}$-condition$\rangle$]$^+$}
6. $\langle K_{ai}$-condition$\rangle$::= {[physicochemical-requirements]$^+$}
7. $\langle$physical-requirements$\rangle$::= (IS-ABSTRACTED-BY $\langle$physicochemical-requirements$\rangle$)

Line 1 indicates that the definition of K_{ai} depends on the following four modeling objects: $\langle K_{ai}$-operator$\rangle$, $\langle K_{ai}$-input$\rangle$, $\langle K_{ai}$-enabling-cond$\rangle$, and $\langle$output$\rangle$. In turn, $\langle K_{ai}$-operator$\rangle$ is defined by a set of attributes (line 2), and $\langle K_{ai}$-input$\rangle$ is defined by a list of $\langle$reaction-centers$\rangle$ (see line 3) with each element of the list defined as an instance of the basic modeling element, `atom`.

Continuing with our task, we recognize that `covalent-bond` requires the specification of $\langle K_{ai}$-enabling-cond$\rangle$, $\langle K_{ai}$-input$\rangle$, and $\langle$output$\rangle$. We will focus on $\langle$output$\rangle$ and $\langle K_{ai}$-enabling-cond$\rangle$ since the specification of $\langle K_{ai}$-input$\rangle$ is transparent (i.e., a list of one or more reaction centers).

The $\langle$output$\rangle$ of `covalent-bond` is an object that represents a covalent bond between two reaction centers. This object (i.e., instance of `covalent-bond`) is not a single (independent) entity. It knows membership and has pointers that connect it to the new `abc`, of which it is a part. The new `abc` recognizes it as a covalent bond. The situation of a nonnil $\langle$output$\rangle$ being returned is dependent on the properties of the reaction centers and level of sophistication encoded into $\langle K_{ai}$-enabling-cond$\rangle$.

The new abc recognizes it as a covalent bond. The situation of a nonnil ⟨output⟩ being returned is dependent on the properties of the reaction centers and level of sophistication encoded into ⟨K_{ai}-enabling-cond⟩.

Specific criteria must be satisfied for a covalent bond to be formed between two atoms. These criteria are based on the physical laws of nature and tempered by our knowledge of the bonding system. They are encoded into the procedure ⟨K_{ai}-enabling-cond⟩.

The general algorithm for ⟨K_{ai}-enabling-cond⟩ is

```
⟨K_ai-enabling-cond⟩:
    input
    initialize
    if  feasible-p
        then true
        return
    end
    return
```

where *feasible-p* represents a list of predicates procedures specifying the criteria for feasibility. The pseudocode procedural form of ⟨K_{ai}-enabling-cond⟩ for the modeling class, covalent-bond, is given below:

```
⟨covalent-bond-enabling-cond⟩
      input: reaction-center-pair
      initialize
            reaction-center-1  ←  first element of reaction-center-pair
            reaction-center-2  ←  last element of reaction-center-pair
      if  favorable-interatomic-distance-p and
          potentially-stable-bond-formation-p and
          available-bonding-electron-p and
          proper-orbital-symmetry-p and
          conservation-laws-upheld-p
          then true
          return
      end
      return
```

The feasibility criteria, following the "if" construct, are implemented as predicate methods. They perform the following evaluations: the method *favorable-interatomic-distance*-p assesses the interatomic distances between the two reaction centers to determine whether bond formation is likely; the method *potentially-stable-bond-formation*-p verifies that the steric repulsion between the reactions centers is less than the potential energy of the system and that the entropic term for bond formation is less than the enthalpy term for bond formation; *available-bonding-electron*-p checks the availability of bonding electrons; *proper-orbital-symmetry*-p determines whether the bonding orbitals are in phase; and *conservation-law-upheld*-p validates whether mass, energy, and momentum have been conserved. These methods are associated with `covalent-bond` using the semantic relationship ***is-method-of***, as we have shown previously, specifically:

favorable-interatomic-distance-p	***is-method-of***	`covalent-bond`
potentially-stable-bond-formation-p	***is-method-of***	`covalent-bond`
available-bonding-electron-p	***is-method-of***	`covalent-bond`

Notice that the method *conservation-laws-upheld*-p has not been associated with `covalent-bond`. This method has been associated with K_{ai}, allowing it to be accessed by all subclass modeling elements of K_{ai}.

We can also specialize these methods according to physical requirements and chemical requirements. This provides an abstraction barrier between what is true (i.e., the physical laws) and what is believed to be true (i.e., current theory). For example, suppose that the methods *favorable-interatomic-distance*-p and *potentially-stable-bond-formation*-p are physical requirements whereas the methods *available-bonding-electron*-p and *proper-orbital-symmetry*-p are chemical requirements. We represent this to `covalent-bond` by associating the top-level methods (i.e., methods associated with K_{ai}), chemical requirements, and physical requirements, to `covalent-bond` directly (i.e., we override the method inherited by the mother model class). This is accomplished using the semantic relationship **is-method-of**, as was demonstrated earlier:

physical-requirement	***is-method-of***	`covalent-bond`
chemical-requirement	***is-method-of***	`covalent-bond`

We then tailor these methods by associating the methods of `covalent-bond` to them. This is achieved using the semantic relationship ***is-composed-of***:

physical-requirement	***is-composed-of***	(favorable-interatomic-distance-p, potentially-stable-bond-formation-p)
chemical-requirement	***is-composed-of***	(available-bonding-electron-p, proper-orbital-symmetry-p)

4. *Extension of the* `composite-operator` *Hierarchical Tree*

Suppose now that we wish to add an additional operator to the model class `composite-operator`, K. For example, consider the addition of radical-disproportionation, a transformation that captures transformations among radicals:

$$\begin{aligned} \cdot C_2H_5 + \cdot C_2H_5 &\longrightarrow C_2H_4 + C_3H_8 \\ &\longrightarrow C_2H_6 + C_3H_8. \end{aligned}$$

As before, the addition of the subclass, $K_{\texttt{radical-disproportionation}}$, to the model class hierarchy requires that links be established between `composite-operator` and `bimolecular-operator`, `bimolecular-operator` and `radical-operator`, and `radical-operator` and `radical-disproportionation`:

bimolecular-operator	*is-a*	composite-operator
radical-operator	*is-a*	bimolecular-operator
radical-disproportionation	*is-a*	radical-operator

Like $K_{\texttt{ai}}$, the modeling element K performs transformations. This necessitates that procedures be developed and associated with K. Unlike $K_{\texttt{ai}}$, the procedures of K are based on known transformations. To identify the procedures required of $K_{\texttt{radical-disproportionation}}$, we return to the BNF syntax presented in Section II.D and reproduce the following links:

1. ⟨composite-operator⟩::= ⟨inputs⟩⟨enabling-conditions⟩
   ⟨ab-initio-operator⟩⟨output⟩
   [⟨composite-operator⟩]
2. ⟨input⟩::= ⟨react-cond⟩⟨abc⟩⟨cb⟩.
3. ⟨output⟩::= ⟨reaction⟩,
4. ⟨abc⟩::= {[(instance-of abc)]$^+$}.
5. ⟨cb⟩::= {[(instance-of cb)]$^+$}.
6. ⟨reaction-cond⟩::= {(instance-of reaction-conditions)}.
7. ⟨enabling-conditions⟩::= {[⟨conditionals⟩]$^+$}.
8. ⟨conditionals⟩::= [[[(instance-of user preferences)]$^+$
   {[(instance-of physical requirements)]$^+$}]].

The syntax of K identifies the elements of any subclass modeling element (e.g., $K_{\texttt{radical-disproportionation}}$). These are elements ⟨inputs⟩ ⟨enabling-conditions⟩ ⟨ab-initio-operator⟩ ⟨output⟩. Each of these is defined by simpler entities, as shown above. (The decomposition of ⟨ab-initio-operator⟩ was given in the previous section.)

In addition, we have seen that a composite operator constructs new reactions through the execution of three methods, K_f, $K_{\text{get-sites}}$, and K_t. In the context of radical disproportionation reactions:

$$R_1 + R_2 \rightarrow R'_1 + R'_2,$$

these three procedures do the following:

1. K_f evaluates the feasibility of radical disproportionation (e.g., R_1 and R_2 are both radicals; energy in the reaction environment is less than that required to cleave the newly formed bond; relative stability of products, etc.)
2. $K_{\text{get-sites}}$ identifies potential reaction sites in R_1 and R_2.
3. K_t supplies a sequence of ab initio operators and applies these operators to the reaction sites identified by $K_{\text{get-sites}}$ to obtain the desired transformation.

The feasibility predicate, K_f, for $K_{\texttt{radical-disproportionation}}$ is similar in spirit to $\langle K_{\text{ai-enabling-cond}} \rangle$ with the exception that the user can encode personal preferences. For example, suppose for safety reasons that the user's interest is focused on a substituted product. This preference could be encoded into K_f by passing in a predicate test, e.g., *substituted-p*, that would be applied to reaction-centers.

```
METHOD: $K_f$
    input: substrate-list
    when substrates are radicals
            collect radical centers into potential-reaction-centers
            evaluate potential-bonds between potential-reaction-centers
                    for each bond in potential-bonds evaluate bond-energy
                            when bond is stable in reaction-environment
                            and user-preferences satisfied
                                    collect potential-reaction-center-pair
                                            into potential-reaction sites
                                    return potential-reaction-sites
                            return
                    return
    end
    return
```

Notice the usage of inputs (i.e., ⟨abc⟩, ⟨cb⟩, and ⟨reaction-condition⟩) in K_f. Substrates and substrate-list are sets of ⟨abc⟩'s; radicals are identified using ⟨cb⟩; bond stability is assessed in the context of ⟨reaction-condition⟩ (i.e., the values of the instance of `reaction-environment`). Additionally note that multiple products may be found since there may be several favorable sites; potential-reaction-sites are collected on the basis of thermodynamic stability relative to the reaction-environment.

The method $K_{\text{get-sites}}$ is responsible for producing products, given a set of potential reaction sites. It applies K_t to potential reaction center combinations and assesses the relative stabilities of the products. The procedure for $K_{\text{get-sites}}$ is given below:

```
METHOD: $K_{\text{get-sites}}$
    input: potential-reaction-sites
    for each reaction-center-pair in potential-reaction-sites
            apply $K_t$ to reaction-center-pair
            collect potential-structure-pair into potential-structures
            return potential-structures
    sort potential-structure-pair by stability
    when stability-acceptable-p of potential-structure-pair
            collect potential-structure-pair into product-pairs
            return product-pairs
    end
    return
```

The call to K_t enables the desired transformation. This transformation (encoded below by K_t) begins by cleaving the bond connecting a mobile moiety (e.g., H) to a parent atom adjacent to reaction-center-1. Products are then produced by creating new bonds between these centers using generic methods (e.g., *make-covalent-bond* and *cleave-bond*) to achieve bond formation and bond cleavage. K_t uses these generic methods to access knowledge contained in $\mathbf{K}_{ai}$.

```
METHOD: $K_t$
    input: reaction-center-pair
    initialize  reaction-center ← first element of reaction-center-pair
                reaction-center ← second element of reaction-center-pair
    cleave-bond mobile moiety adjacent to reaction-center-1
    biradical ← reaction-center-1 structure
    product-1 ← apply make-covalent-bond to mobile moiety
                and reaction-center-2
    product-2 ← apply make-covalent-bond to biradical
    end
    return
```

By changing the transformation code and by adding selector functions we can enhance K_t to enable additional disproportionations. These may include transformations depicting disproportionation of such radicals as R_3CO_2 and R_2HCO_2.

Finally, we associate the procedures K_f, K_t, and $K_{\text{get-sites}}$ to the modeling class $\mathbf{K}_{\textbf{radical-disproportionation}}$ using the semantic relation **is-method-of**:

K_f	***is-method-of***	radical-disproportionation
$K_{\text{get-sites}}$	***is-method-of***	radical-disproportionation
K_{ft}	***is-method-of***	radical-disproportionation

We also establish a second (redundant) link using the semantic relationship ***is-described-by***. In addition, LCR establishes automatically the following three semantic links, which are the symmetrical relationships of the preceding three:

radical-disproportionation	***is-described-by***	K_t
radical-disproportionation	***is-described-by***	$K_{\text{get-sites}}$
radical-disproportionation	***is-described-by***	K_f

Now, the modeling class, $\mathbf{K}_{\textbf{radical-disproportionation}}$, is fully recognized and usable by **K**. An evaluation of $\mathbf{K}_{\textbf{radical-disproportionation}}$ on R_1 and R_2 will return an instance of the class `reaction` or set of instances (reflecting multiple favored products). Each `reaction` constructed will contain values for the attributes (e.g., reactants, products, stoichiometry, reaction-environment, enabling conditions) as specified by the modeling element `reaction` (see Section II.A).

B. The "Model Class Decomposition Digraph" (MCDD)

As the examples of the previous section have indicated, the modeling objects in LCR's hierarchies are built from simpler ones by linking them into composite entities. The mechanism for the linkage is the declaration of the semantic relationship between two modeling entities. Thus, starting from one of the nine basic modeling classes (the mother-class), we can create a modeling subclass and endow it with new specialized attributes and methods. Each new attribute could be modeled as an instance of an existing or a new modeling class. This mechanism can be applied recursively until desired representation has been achieved. The assumptions that guide the subclassing of a new modeling element from an existing one, and the specification of its attributes form the scope of the *modeling context*.

Let a modeling object be represented by a node, and let an *edge* represent its semantic link to another modeling object (i.e., another node). Then, the gradual and modular definition of a model can be represented by an evolving directed graph, which is called *model class decomposition digraph* (MCDD). The MCDD represents the net result of the model class decomposition taking into account all the components created at that time. The basic idea is simple and is schematically depicted in Fig. 10. At state 1, the model root A has only one constituent entity called B. At state 2, the entity C has been incorporated into the substructure of B; but C is not a simple entity, it is a composite object itself (e.g., a covalent bond ***is-described-by*** a set of mathematical components). Consequently, its whole structure is pasted into B's structure. Stage 3 shows that B's substructure has been further specified by incorporating an additional composite object called D. Following with B's definition, stages 4–6 show that a new constitutive part, named E, is being gradually created. This presents a contrast with the previous incorporation of entities C and D; they had predefined substructures. This process will continue until A's structure has been completely specified.

The modularity of the model class definition process allows the modeler to make use of preexisting classes or to gradually define new ones. This feature combined with the communicative properties of the semantic relations makes possible the modification or upgrading of a model class with the minimum of effort.

MCDDs are made up of nodes and edges. The digraph's *nodes* have the particularity of being classes; each one of them corresponds to a constituent entity of the composite object that the model class is representing at the time. Pairs of nodes in a digraph are linked by *edges*. In a MCDD edges describe the semantic relations that victualed the entities represented by the nodes. Any edge of a MCDD can be associated with any of the following specification semantic relations: ***is-composed-of, is-connected-by***, and ***is-described-by***. For example,

A MCDD is a digraph $G = (X, U_R)$, where

(a) $X = \{x_1, x_2, \ldots, x_n\}$ is the set of nodes representing classes of modeling objects.

(b) $U_R = \{(x_i, x_j) \in X \times X / x_i R x_j\}$ is the set of edges connecting nodes x_i and x_j through the three specification oriented semantic relationships, mentioned above.

A MCDD can be characterized as an acyclic digraph whose starting node, S, has a nil predecessor [i.e., $\Gamma^-(S) = 0$] labeled with the name of the system the model is intended to represent. The successor nodes of S, $\Gamma^+(S)$, are labeled with the names of the first-level components of the model. Each of these first-level constituent entities may itself be decomposed into second-level entities, which will be represented by new successor nodes in the digraph, i.e., $\Gamma^+[\Gamma^+(S)]$. Thus, the model of any node, x_i, is the

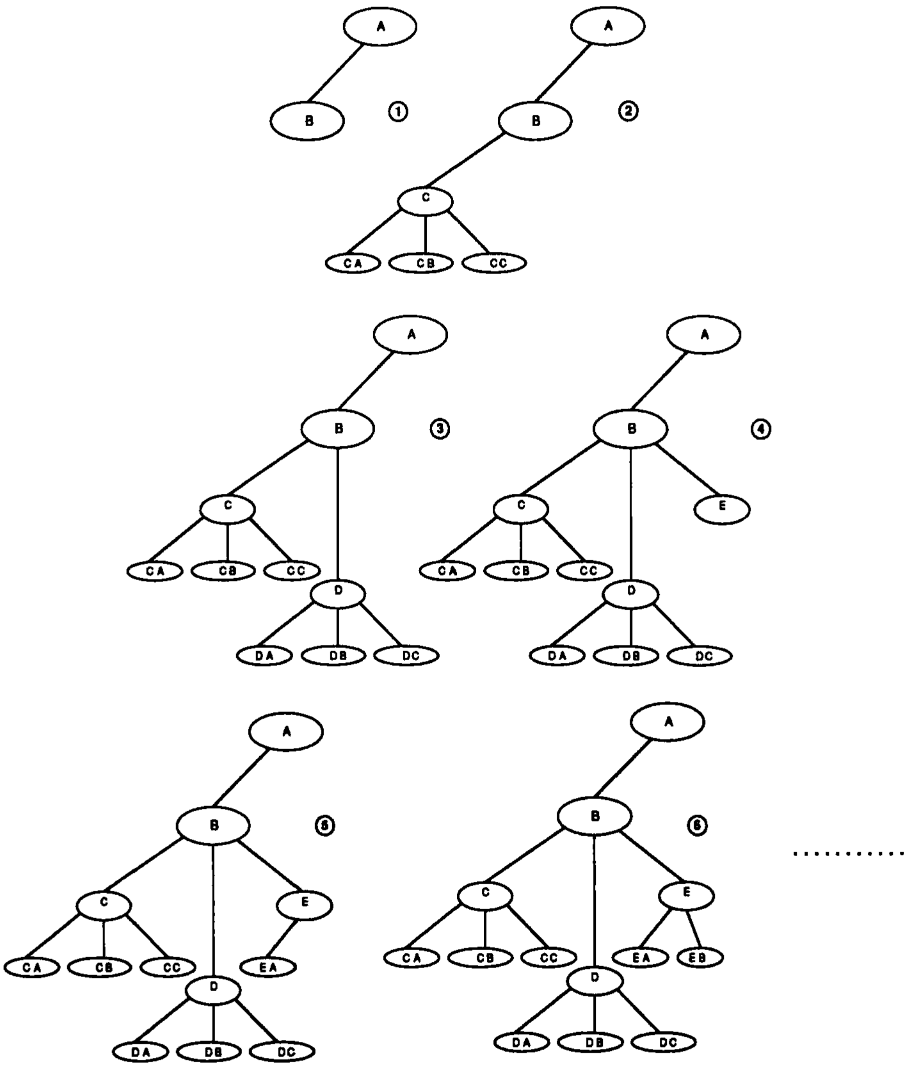


FIG. 10. The evolution of an MCDD's structure. (Reprinted from Stephanopoulos, G., Henning, G., and Leone, H. MODEL.LA A modeling language for process engineering. Part I, *Comp. Chem. Eng.* **14**, Page 813, Copyright 1990, with kind permission from Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

subgraph of the MCDD generated by x_i and its descendants of $(\Gamma^+(x_i) \cup \{x_i\})$;

$$\mathrm{MCDD}(x_i) = \left(\left\{\hat{\Gamma}^+_{\mathrm{MCDD}}(x_i) \cup \{x_i\}\right\}, U_{\mathrm{R}}(x_i)\right),$$

where $\hat{\Gamma}^+_{\mathrm{MCDD}}(x_i)$ is the set of descendants of x_i in MCDD and $U_{\mathrm{R}}(x_i)$ is given by

$$U_{\mathrm{R}}(x_i) = \left\{(x_{\mathrm{a}}, x_{\mathrm{b}}) \in \left\{\hat{\Gamma}^+_{\mathrm{MCDD}}(x_i) \cup \{x_i\}\right\} X \left\{\hat{\Gamma}^+_{\mathrm{MCDD}}(x_i) \cup \{x_i\}\right\} / x_{\mathrm{a}} R x_{\mathrm{b}}\right\}.$$

Formally, a MCDD represents a hierarchical decomposition of a model into constituent entities. Since this decomposition must ultimately terminate, a MCDD should not have directed circuits. Thus, MCDDs are digraphs that have the following properties:

Strict hierarchy. No component node can appear in the path that has as initial end-point one of its constituent entity nodes. Consequently, no component can have a decomposition which eventually contains a constituent entity of the same type.

Uniformity. Any two nodes with the same label have attached to them the same constituent entities. This property implies that any two components of the same type have the same isomorphic decomposition structures.

These properties are particularly important because they guarantee the development of a reasonable model and easy access to its different views, components, etc. An additional property, resulting from the uniformity axiom is that all MCDDs have the capability to specify the digraphs that are associated with the instances of the model class. The instantiation of a MCDD generates a *model decomposition digraph* (MCD). MCDs, like MCDDs, do not have directed circuits. They are finite graphs, obtained as a result of the instantiation of another finite graph.

C. Generation and Representation of Reaction Pathways

In the previous sections we discussed how we can use LCR's mechanisms to add new modeling elements, or to generate representations from existing modeling elements. In this section we will show how to use LCR for the construction of reaction pathways. We will employ examples from the domain of the *free-radical chemistry*, such as oxidation of butane.

LCR provides a method for generating all possible reactions and identifying the resulting pathways, given a set of substrates and a reaction environment. The keyword arguments (i.e., arguments with a colon prefix) accepted by this method are given below:

```
(find-all-pathways :substrates :operators :override-environment
                   :initiator-p)
```

The method tests for an override reaction environment allowing the user to investigate the influence of various environments without manually resetting the environment attribute of each substrate. [Recall that an `abc` (i.e., a substrate) "knows" its environment.] The user may also identify whether an initiator is present or believed to be present. This allows the user to investigate the effect of various reaction trajectories without knowing specifics concerning the initiator mechanism. The keyword *:operations* allows the user to specify the types of transformation to be used; the default value is `composite-operator`. The user may specify other operators as well. This focuses the elucidation of pathways. For example, by supplying $K_{\texttt{free-radical}}$ to the keyword argument the user can investigate pathways of free-radical origin.

Alternatively, the user may wish to investigate all theoretically feasible pathways subject to a set of preferences. This is accomplished by supplying *:operators* with K^*, a modified composite operator, which has encoded the user preferences. Recall that K^* as an instance of the `composite-operator`, generates reactions by executing the methods K_t, K_f, and $K_{\text{get-sites}}$. K^* contains no encoded transformations within K_t but rather applies the complete set of instances of ab-initio-operator (i.e., $K_{\texttt{ai}}$) directly. The K_f procedure reflects user preferences. This allows the user to investigate pathways having prespecified features ($\Delta G < 10$ kcal/mol, stereocenters, etc.). Generation of these pathways is accomplished as follows:

```
(find-all-pathways :substrates (butane oxygen) :operators K*
                   :initiator-p nil)
```

Similarly, if there are no preferences, theoretically feasible reactions can be generated by calling find-all-pathways with *:operators* $K_{\texttt{ab-initio}}$ directly:

```
(find-all-pathways :substrates (butane oxygen) :operators K_ab-initio
                   :initiator-p nil)
```

This functionality allows the user to control the scope for the identification of potential pathways. Pathways connecting two states are identified using the method *find-all-pathways-connecting*. Its general form is

```
(find-all-pathways-connecting :substrates   :desired-products
                                            :operators   :initiator-p
                                            :strategy)
```

where substrates may be a class allowing the user to investigate pathways leading from a desired product to a class or classes of substrates. Alternatively, *:substrates* can be left unspecified. Once a control (e.g., synthesis) strategy has been specified, the method, *find-all-connecting-pathways*, chains on K_f and K_t methods of K, using the semantic relation ***is-described-by***, and appropriate selector functions. The current implementation of LCR allows chaining only in the forward direction (i.e., synthetic direction) when $\mathrm{K}_{\texttt{ab-initio}}$ or K^* is supplied to *:operators*.

Let us return now to the oxidation of butane. We will illustrate the generation of feasible pathways using K^*. In this example, K^* was restricted to thermodynamically viable free-radical pathways; products were restricted to their homologous series. The generation call was

```
(find-all-pathways  :substrates (butane oxygen)  :operators K*
                    :initiator true)
```

Figure 11a presents several of the pathways identified when *s*-BuOH is formed during the initiation process. The attributes of the initiation reaction, `initiation-1`, for the oxidation of butane are presented in Fig. 12a, where objects are denoted by #⟨object-name⟩. Note that each of these objects can in turn be expanded or disaggregated using the semantic relationships. Figure 12b illustrates one of the many pathways, pathway-1, generated during the oxidation of butane into *s*-BuOH and AcEt. The expansion of the object, #⟨initiation-1⟩, an element in the value of the attribute, "composing-reactions," results in the description shown in Fig. 12a. The user has access to this information at every step of the synthetic process using the semantic relationships provided by LCR (see Section III.E).

By changing the values of the object, `reaction-environment`, one can generate new trajectories (denoted in Fig. 11b by dashed lines), corresponding to a higher-energy environment. Figure 11c presents the reaction pathways resulting from yet another `reaction-environment`. The various pathways stemming from these environments are combined in Fig. 13. The assumption set and conditions specifying each environment are managed by the context generation utility of LCR, discussed in the next section.

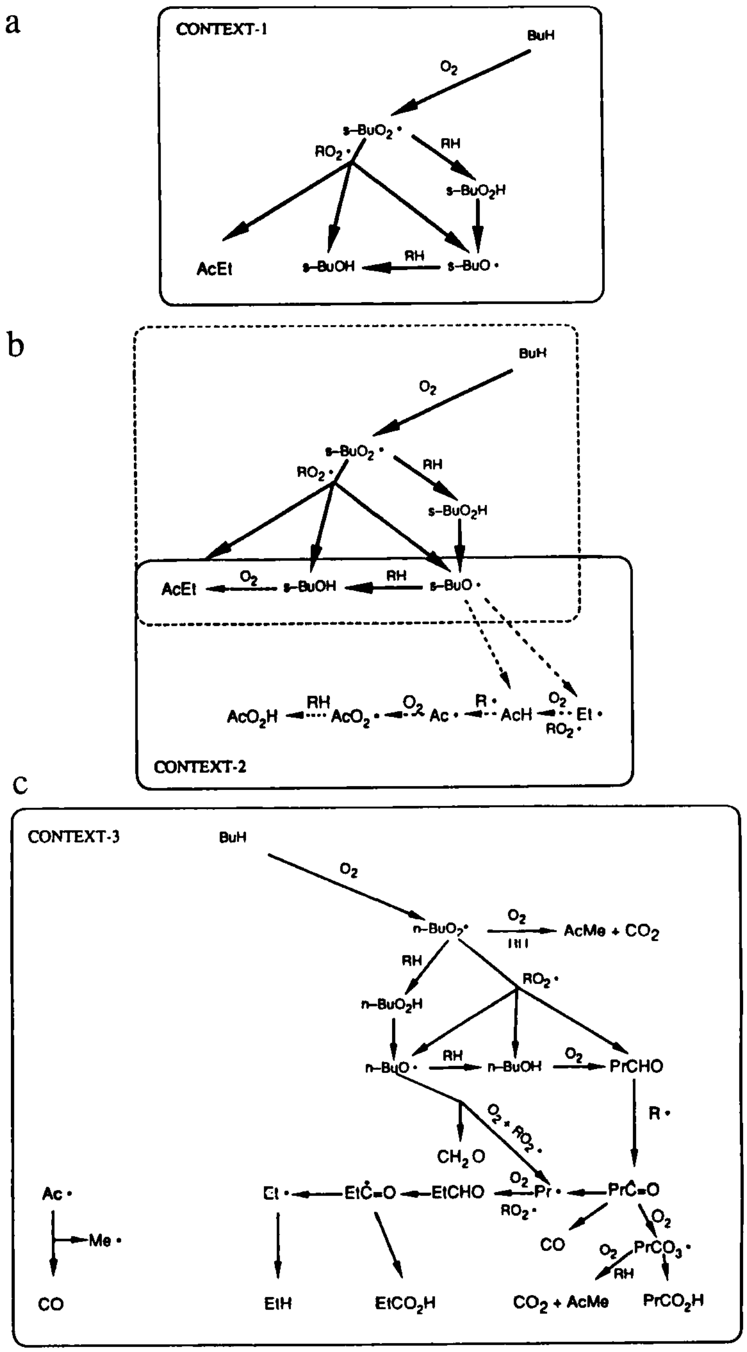


FIG. 11. Three distinct abstractions in describing the oxidation of butane.

a

Reaction Object	
identifier:	*unbound*
name:	initiation-1
reactants:	(#<$C_4H_{10}O_2$>)
products:	(#<C_4H_9O> #<HO>)
stoichiometry:	((#<$C_4H_{10}O_2$>. -1) (#<C_4H_9O>. +1) (#<HO>. +1))
reaction-environment:	#<reaction-environment-1>
enabling-conditions:	K_f
composing-transformations:	K_t
composing-reactions:	*unbound*
rate-expression:	#<rate-expression-1>
equilibrium-constant:	#<equilibrium-constant-1>
context:	#<context-1>

b

Pathway Object	
identifier	*unbound*
name	pathway-1
reactants	(# <C_4H_{10}> #<O_2>)
products	(#<$CH_3COCH_2CH_3$>)
stoichiometry	unbound
competing-pathways	(<pathway-2> #<pathway-3>)
composing-reactions	(#<initiation-1> #<initiation-2> #<abstraction-1> #<disproportionation-1> ...)
global-rate-expression	#<composite-rate-exp-1>
global-equilibrium-constant	*unbound*

FIG. 12. The description of (a) a reaction object and (b) a pathway object during the oxidation of butane.

By indexing and storing new reactions in their entirety, i.e., by storing the complete objects making up reactions, chemical structures, and chemical conditions, whether they were generated by K, or discovered through the use of K* or K_{ai}, we have given LCR the ability to learn. Since a context is associated with each reaction, retrieval of specific reactions (from the ever-expanding library of chemical reactions) and incorporation in the evolving generation of pathways can be focused to the desired context, in order to suit specific needs.

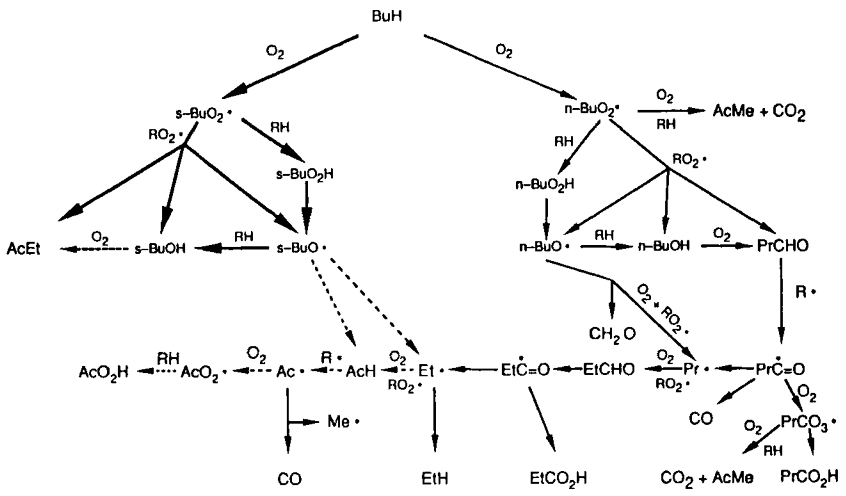


FIG. 13. Linking distinct contexts used for the representation of butane oxidation.

D. Creation of Contextual Reaction Models

The generation and representation of molecular structures, reaction, and pathways is always done within the scope of some "context." In Fig. 11 we see three (3) distinct networks of reactions, occurring during the oxidation of butane, and corresponding to three (3) distinct "contexts" of operating conditions. In general, the representation of any object within LCR depends on a set of assumptions that can be classified into the following three categories:

1. *Assumptions on structural components.* They declare component/subcomponent relations or level combinations of subcomponents: Chemical A has been conjectured as a `cyclic-aliphatic` molecule; bond between atoms A_1 and A_2 is assumed to be `covalent-bond`; pathways `P` is assumed to be a serial concatenation of pathways, P_1, P_2, and P_3.
2. *Assumptions on behavior or functionality.* These are relations defining: reactivity of various reaction centers in a molecule, heats of reaction, bond strength between two atoms, etc.
3. *Assumptions on the values of variables*, describing attributes of molecular structures or/and reactions.

These assumptions are expressed in terms of generalized constraints, which represent Boolean, qualitative, semiquantitative, or quantitative relationships among the modeling elements. The set of generalized constraints forms the "*context*" within which a model is valid, and are viewed as hard constraints taking on a Boolean character; thus, they are either satisfied or violated (never satisfied to a certain degree).

Let us assume that a model X developed within CONTEXT-1, is to be modified. CONTEXT-2 is created as a "child" of CONTEXT-1, that will be referred to as the "parent-context." In principle, CONTEXT-2 "inherits" all the assumptions that are valid in CONTEXT-1. Then, in order to account for the modifications of model X it is possible to change the *inherited assumptions with additions and deletions*. The changes are covered by the following rule:

> All assumptions in CONTEXT-1 are "inherited" by CONTEXT-2 unless overridden in CONTEXT-2.

Generalizing this rule, we can say that:

> The assumptions true in a given context are all the assumptions that are true in the parent-context, minus the assumptions that have been specifically deleted in it, plus any assumption that has been specifically added in it.

Following with this hypothetical case, let us assume that a different representation of X is required. Now, two options are given to us: (1) if the new representation is seen as an alternative to the one already introduced in CONTEXT-2, a new context called CONTEXT-3, should be created as a child of CONTEXT-1; (2) If the modification is intended to change model X built in CONTEXT-2, CONTEXT-3 should be created as a child of CONTEXT-2. The first choice is analogous to the situation found in butane oxidation. The assumptions corresponding to the free-radical oxidation of butane are encapsulated in CONTEXT-1 (Fig. 11a). Given that two distinct global pathways have to be analyzed, CONTEXT-1 gives rise to CONTEXT-2 and CONTEXT-3, which encapsulate the assumptions that generate the representations of Fig. 11b and 11c, respectively. CONTEXT-2 and CONTEXT-3 siblings both have information in common with CONTEXT-1, but there is no direct relation between them. The second situation corresponds to many multistep syntheses. During the synthesis of a complex molecule the chemist makes assumptions regarding how the molecule should be constructed, and these assumptions give rise to new assumptions. For example, assumptions regarding the specification of a carbon skeleton with correct key group orientation may be kept in CONTEXT-A. This context may give rise to CONTEXT-B placing restrictions (i.e. assumptions) on the latent functionality of the skeleton. These restrictions, in turn, may give rise to CONTEXT-C, i.e., assumptions regarding the need for certain protective groups.

Generalizing, we can state that contexts are always arranged in hierarchical manner. The relations among them are established in a direct graph, generated by the context relation (CR) on the set of contexts, C. The relation CR is read as "*gives-rise-to*," and the digraph is called MODEL ABSTRACTION TREE (MAT). It is a tree because each node representing a context has only one predecessor. A MAT is defined by

$$\text{MAT} = (C, U_{\text{R}}),$$

where

$$C = \{\text{CONTEXTS}\},$$

$$U_{\text{R}} = \{(C_i, C_j) \in C \times C / C_i CR C_j\}.$$

This tree of contexts is similar to the "class precedence list" used in hierarchical inheritance mechanisms. An example of such tree is presented in Fig. 14, where CONTEXT-1 gives rise to CONTEXT-2 and CONTEXT-3, and CONTEXT-2 gives rise to CONTEXT-4. All contexts represent different reaction pathways for the same overall reaction.

1. Communication between Contextual Models

Contexts can be related as (a) siblings (e.g., CONTEXT-2 and CONTEXT-3 in Fig. 14) or (b) "parent"–"child" (e.g., CONTEXT-1 and CONTEXT-2 in Fig. 14).

In the first case, the contexts do not need to communicate with each other, but in the second case, information must pass from the parent-context to the child-context. The operations associated with the modification of the parent-context to produce the child-context, presently available in LCR are

1. *MODIFY*: an operation applied to an individual assumption in order to alter its value.
2. *DELETE*: an operation applied to a single assumption in order to alter its value.
3. *ABSTRACT*: an operation that is applicable to the set of assumptions representing an object model. The representation of this object is abstracted to obtain a less detailed description of its structure.
4. *DISAGGREGATE*: is the opposite to ABSTRACT. It is applied to the assumption set, representing a model, but its purpose is to introduce more refinement.

Whereas MODIFY and DELETE are simple operations, ABSTRACT and DISAGGREGATE are quite complex. Whichever of the last two operations is performed during the modification of a new context, it should allow the user to bind together components that are conceptually related in the two contexts. Thus, the ABSTRACT and

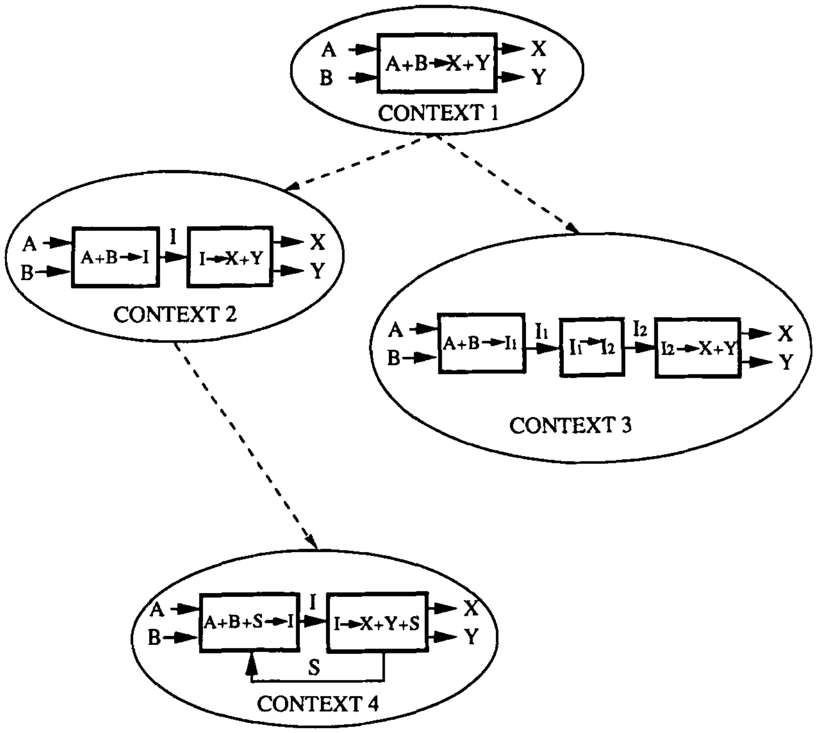


FIG. 14. A tree of contextual inheritances.

DISAGGREGATE operations should provide the framework for the communication across contexts. The symmetric operations of abstraction and disaggregation can be described by an algorithm that considers them in a very general sense, and can be summarized as follows:

```
ABSTRACTION (NEW-CONTEXT NEW-UNIT OLD-UNITS)
    ESTABLISHED-STRUCTURAL-COMPATIBILITY
    (NEW-CONTEXT NEW UNIT OLD-UNITS)
    ESTABLISH-BEHAVIORAL-COMPATIBILITY
    (NEW-CONTEXT NEW-UNIT OLD-UNITS)
END
DISAGGREGATION (NEW-CONTEXT NEW-UNITS OLD-UNIT)
    ESTABLISH-STRUCTURAL-COMPATIBILITY
    (NEW-CONTEXT NEW-UNITS OLD-UNIT)
    ESTABLISH-BEHAVIORAL-COMPATIBILITY
    (NEW-CONTEXT NEW-UNITS OLD-UNIT)
END
```

Since contexts have only one parent there is no need to specify the name of the context where the old-units are located. With the name of a new context, its parent context will be uniquely specified. The details of the above algorithms will be discussed in the following paragraphs, where we will make extensive use of the semantic links, ***is-disaggregated-in*** and ***is-abstracted-by***.

2. *Structural Compatibility among Contextual Models*

Structural compatibility analysis operates on pathways that are located in two different contexts, bound together by a CR relation. During the disaggregation operation, a pathway that exists in the parent context is substituted by a set of pathways in the next context. The structural compatibility operation will establish the following semantic link:

P_{old} ***is-disaggregated-in*** (SET.OF($P_1, \ldots, P_n,$)) in NEW-CONTEXT

Since a parent-context may have several children, it is always necessary to specify the name of the child-context when the disaggregation occurs. For example, during the oxidation of butane, the context `butane-oxidation-reaction-environment` gives rise to the contexts `butane-oxidation-termination-1` and `butane-oxidation-termination-2`. So, the relation

`butane-pathways` ***is-disaggregated-in***
(SET.OF `(initiation, propagation, termination))`
in `butane-oxidation-termination-1`

provides an unambiguous link and avoids any confusion with models located in the other contexts (`initiation, propagation`,...,etc.)

The abstraction procedure is just the inverse operation:

(SET.OF ($P_1, \ldots, P_n,$)) ***is-abstracted-by*** P_{new} in NEW-CONTEXT

Now, a set of `P` values are abstracted in a single pathway. Figure 15a shows an example of the abstraction and disaggregation processes between CONTEXT-i and CONTEXT-j. If CONTEXT-j is created as a child of CONTEXT-i, the structural-compatibility operation will generate the link

P ***is-disaggregated-in*** (SET.OF (P_1 P_2)) in CONTEXT-j

On the other hand, if CONTEXT-i is a child of CONTEXT-j, the structural compatibility operation would generate the following link:

(SET.OF (P_1, P_2)) ***is-abstracted-by*** P in CONTEXT-i

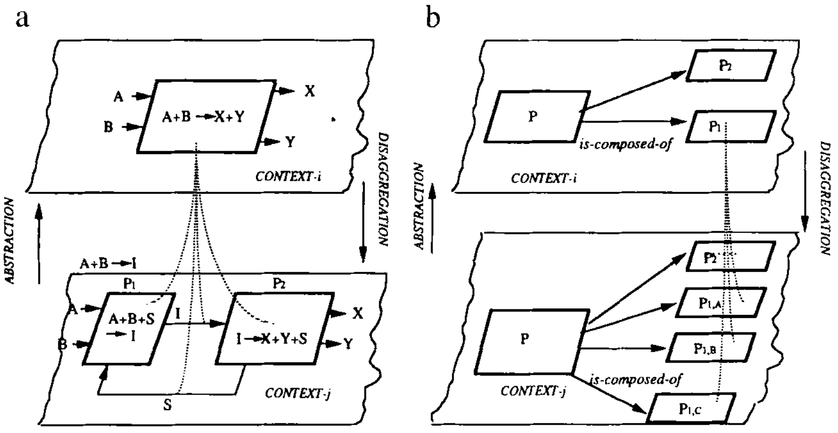


FIG. 15. Aggregation and disaggregation of representations of reacting systems.

However, there are more complex situations that need further analysis. Specifically, the case of pathways that are being disaggregated or abstracted and that themselves are not simple pathways from a structural point of view, but are also "part-of" other pathways in the parent context. What happens with these semantic links in the new context? A typical situation is depicted in Fig. 15b, where pathway P_1, which is *part-of* pathway P in CONTEXT-i, is disaggregated in pathways $P_{1,A}$, $P_{1,B}$ and $P_{1,C}$ in CONTEXT-j. The structural-compatibility procedure, apart from establishing ***is-disaggregated-in*** links across the contexts, should also establish ***is-part-of*** links between pathways $P_{1,A}$, $P_{1,B}$ and $P_{1,C}$, and pathway P.

More formally, the structural-compatibility procedure should perform the following checking operation every time a pathway P_{old} is disaggregated in a new context.

IF P_{old} *is-part-of* ?PATHWAY-X in OLD-CONTEXT
THEN ?PATHWAY-Y *is-part-of* ?PATHWAY-X in NEW-CONTEXT

where ?PATHWAY-Y is one of the pathways in which P_{old} is disaggregated, i.e., the following link exists between them:

P_{old} ***is-disaggregated-in*** ?PATHWAY-Y in NEW-CONTEXT

In order to avoid the generation of redundant links that the checking operation can produce due to the transitivity axiom, ?PATHWAY-X, will be restricted to be the predecessor of P_{old} that is related to P_{old} by means of a ***is-composed of-*** link. This condition is met by the only member of the following set

$$\{x \mid x \in A \wedge \{\{y \mid x \textit{ is-composed-of } y\} \cap A\} = \phi\}$$

where

$$A = \{x \mid x \textit{ is-composed-of } \mathrm{P_{old}}\}$$

Let us now consider the inverse situation, depicted in Fig. 15b, where pathways $P_{1,A}$, $P_{1,B}$, and $P_{1,C}$, that are part of the pathway P in CONTEXT-j are abstracted in pathway P_1 in CONTEXT-i. In this case, the structural-compatibility procedure will establish ***is-abstracted-by*** links across the contexts and a ***is-composed-of*** link between pathways P and P_1 in CONTEXT-i. In order to consider situations like the one presented above, the following checking operation is also carried out every time a set of pathways $\{\mathrm{P_i},\ i = 1, \ldots, n\}$ ***is-abstracted-by*** the pathway $\mathrm{P_{new}}$ in a new context.

IF ?PATHWAY-X ***is-composed-of*** $\{\mathrm{P_i},\ i = 1, \ldots, n\}$ in OLD-CONTEXT
THEN ?PATHWAY-X ***is-composed-of*** $\mathrm{P_{new}}$ in NEW-CONTEXT

where $\{\mathrm{P_i},\ i = 1, \ldots, n\}$ ***is-abstracted-by*** $\mathrm{P_{new}}$ in NEW-CONTEXT.

For the same reasons expressed above, that is, to avoid the creation of redundant links, ?PATHWAY-X will be restricted to be the pathway that is a predecessor of the members of the set $\{\mathrm{P_i},\ i = 1, \ldots, n\}$, and which is also related to them by a ***is-composed-of*** link. Like before, ?PATHWAY-X is the only member of the set

$$\{x \mid x \in P \wedge \{\{y \mid x \textit{ is-composed-of } y\} \cap P\} = \phi\}$$

where

$$P = \{x \mid x \textit{ is-composed-of } \{\mathrm{P_i},\ i = 1, \ldots, n\}\mathrm{old}\}$$

E. Case Study: Ethane Pyrolysis

To demonstrate the utility of LCR and the interaction between various modeling elements, semantic relationships, and supporting methods, let us first consider the pyrolysis of ethane forming principally ethylene and hydrogen. Although this example is relatively simple, it highlights the functionality of LCR and underscores various issues that require resolution for computer implementation to be successful.

1. Initialization

We begin by initializing the instance of `atom-bond-configuration`, which represents `ethane` (Table IV). Each attribute in Table IV whose value is an object can in turn be expanded, using the semantic relationships of LCR. For example, the attribute "empirical-formula" contains a list of `ab-atom` instances, whose expansion makes accessible those properties of an atom, which are independent of its environment (Table V), and include electronegativity, valence electrons, and atom-weight. Similarly, expansion of the instances `atom` in the attribute "atoms" describes substrate-dependent properties of an atom (Table VI), which include "bonds," "hybridization," "neighbor-atoms," "neighbor-groups," etc. Similarly, Fig. 16 shows the attributes of a particular instance of `bond`, describing the characteristics of a specific bond in ethane, while Table VII shows the attributes describing the group "methyl-1".

2. Generation of Pathways

The identification of various free-radical pathways and underlying elementary reactions involved in the pyrolysis of ethane are evaluated by applying the procedure, FIND-ALL-PATHWAYS, to the substrates, and associating with that call the designated instance of `reaction-environment` and the group of composite operators to be used (e.g., K, K^*, $K_{\text{ab-initio}}$). The call to FIND-ALL-PATHWAYS is shown below:

```
(FIND-ALL-PATHWAYS
    :substrates (ethane) :operators K_free-radical
    :override-environment reaction-environment)
```

The method FIND-ALL-PATHWAYS begins by calling the composite operators that constitute the domain of free-radical operations on the substrates specified by the keyword argument *:substrates*. These operators loop over the substrate list and evaluate properties specified by $K_{\texttt{free-radical}}$. Homolytic dissociation of `ethane` is initiated when FEASIBLE-P, a compound predicate method of $K_{\texttt{initiation}}$ ($K_{\texttt{initiation}}$ is an operation of $K_{\texttt{free-radical}}$), is evaluated on the sites identified by $K_{\text{get-sites}}$. These sites are passed to FEASIBLE-P, which searches the `reaction-environment` to establish potential radical forming processes: thermal cleavage, photochemical cleavage, and oxidation–reduction processes. Attributes describing `reaction-environment` establish thermal cleavage as a likely initiation mechanism; photochemical cleavage

TABLE IV

Attribute Values of an Instance of Ethane

identifier:	"C2H6-T0779"
name:	"Ethane"
atoms:	(#⟨ATOM C-T0764⟩ #⟨ATOM H-T0765⟩ #⟨ATOM H-T0766⟩ #⟨ATOM H-T0767⟩ #⟨ATOM C-T0768⟩ #⟨ATOM H-T0769⟩ #⟨ATOM H-T0770⟩ #⟨ATOM H-T0771⟩)
bonds:	(#⟨BOND b-T0772⟩ #⟨BOND b-T0773⟩ #⟨BOND b-T0774⟩ #⟨BOND b-T0775⟩ #⟨BOND b-T0776⟩ #⟨BOND b-T0777⟩ #⟨BOND b-T0778⟩)
empirical-formula:	((#DB-ATOM 414000575⟩.6) (#⟨DB-ATOM 414001073⟩.2))
empirical-formula-string:	"C2H6"
molecular-weight:	30.07
charge:	0.0
terminal-skeleton-atoms:	(#⟨ATOM C-T0764⟩ #⟨ATOM C-T0768⟩)
equivalent-atoms:	((#⟨ATOM C-T0768⟩ #⟨ATOM C-T0764⟩) (#⟨ATOM H-T0771⟩ #⟨ATOM H-T0765⟩ #⟨ATOM H-T0766⟩ #⟨ATOM H-T0767⟩ #⟨ATOM H-T0769⟩ #⟨ATOM H-T0770⟩))
equivalent-bonds:	((#⟨BOND b-T0775⟩) (#⟨BOND b-T0778⟩ #⟨BOND b-T0772⟩ #⟨BOND b-T0773⟩ #⟨BOND b-T0774⟩ #⟨BOND b-T0776⟩ #⟨BOND b-T0777⟩))
weakest-bond:	(#⟨BOND b-T0775⟩)
weakest-bond-strength:	82.6
weakest-bond-strength-ratio:	1.0
ordered-eq-bonds:	((#⟨BOND b-T0775⟩) (#⟨BOND b-T0778⟩ #⟨BOND b-T0772⟩ #⟨BOND b-T0773⟩ #⟨BOND b-T0774⟩ #⟨BOND b-T0776⟩ #⟨BOND b-T0777⟩))
groups:	(#⟨GROUP methyl-1⟩#⟨GROUP methyl-2⟩ #⟨GROUP terminal-sp3-methylene-1⟩ #⟨GROUP terminal-sp3-methylene-2⟩)
group-bonds:	#⟨BOND b-T0775⟩ #⟨BOND b-T0772⟩ #⟨BOND b-T0776⟩)
meta-groups:	#⟨GROUP ethyl-1⟩ #⟨GROUP ethyl-2⟩)
methyl-carbon:	NIL
primary-carbons:	(#⟨ATOM C-T0764⟩ #⟨ATOM C-T0768⟩)
secondary-carbons:	NIL
tertiary-carbons:	NIL
terminal-carbons:	(#⟨ATOM C-T0764⟩ #⟨ATOM C-T0768⟩)
backbones:	(#⟨ATOM C-T0764⟩ #⟨ATOM C-T0768⟩)
backbone-length:	2
progenitor:	NIL
environment:	#⟨reaction-environment-1 11235723⟩
⋮	⋮

TABLE V

DATA FROM AN EXTERNAL DATABASE FOR `atom` "CARBON"

Name:	"Carbon"
atomic-symbol:	"C"
atomic-number:	6
atom-weight:	12.001
valence:	4
row:	"1"
column	"4a"
orbitals:	*unbound*
valence-electrons:	4
electronegativity:	*unbound*

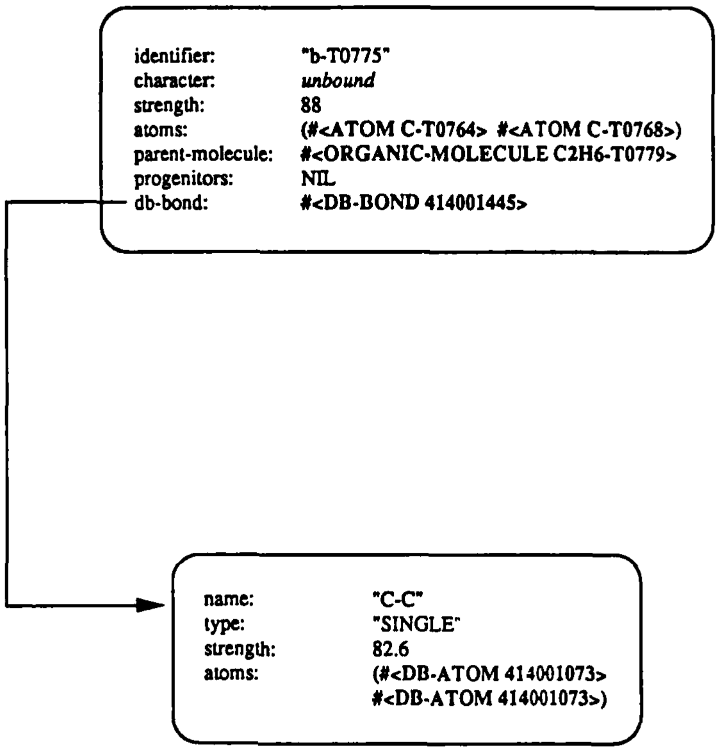
and initiation by oxidation–reduction processes are eliminated because their attribute values are unbound or nil (i.e., nontrue).

Using this knowledge, $\kappa_{\texttt{initiation}}$ constructs a list of potentially-cleavable-bonds by applying the method IDENTIFY-WEAKEST-BONDS to the bond representation of each substrate contained in the substrate-list (i.e., `ethane`). IDENTIFY-WEAKEST-BOND returns a list of bonds,

TABLE VI

SELECT ATTRIBUTES OF AN ATOM

identifier:	"C-T0764"
old-identifier:	NIL
free-valence:	0
diradical-p:	NIL
formal-charge:	0.0
electron-withdrawing-substituent-p:	NIL
connectivity-number:	4
hybridization:	"sp3"
conjugated-p:	NIL
p-orbitals:	NIL
open-approach-p:	T
parent-molecule:	#⟨ORGANIC-MOLECULE C2H6-T0779⟩
progenitors:	NIL
parent-groups:	(#⟨GROUP methyl-1⟩ #⟨GROUP terminal-sp3-methylene-1⟩ #⟨GROUP ethyl-1⟩ #⟨GROUP ethyl-2⟩)
neighbor-atoms:	(#⟨ATOM C-T0768⟩ #⟨ATOM H-T0767⟩ #⟨ATOM H-T0766⟩ #⟨ATOM H-T0765⟩)
neighbor-groups:	(#⟨GROUP methyl-2⟩ #⟨GROUP terminal-sp3-methylene-2⟩)
bonds:	(#⟨BOND b-T0775⟩ #⟨BOND b-T0774⟩ #⟨BOND b-T0773⟩ #⟨BOND b-T0772⟩)
DB-atom:	#DB-ATOM 414001073⟩



```
identifier:        "b-T0775"
character:         unbound
strength:          88
atoms:             (#<ATOM C-T0764> #<ATOM C-T0768>)
parent-molecule:   #<ORGANIC-MOLECULE C2H6-T0779>
progenitors:       NIL
db-bond:           #<DB-BOND 414001445>
```

```
name:              "C-C"
type:              "SINGLE"
strength:          82.6
atoms:             (#<DB-ATOM 414001073>
                   #<DB-ATOM 414001073>)
```

FIG. 16. Description of a bond with reference to data from an external database.

TABLE VII

SELECT ATTRIBUTES OF METHYL GROUP

identifier:	"methyl-1"
string-identifier:	"-CH3"
type:	*unbound*
neighbor-groups:	(#⟨GROUP methyl-2⟩ #⟨GROUP terminal-sp3-methylene-2⟩ #⟨GROUP ethyl-1⟩ #⟨GROUP ethyl-2⟩)
comprising-atoms:	(#⟨ATOM C-T0764⟩ #⟨ATOM H-T0767⟩ #ATOM H-T0766⟩ #⟨ATOM H-T0765⟩)
connecting-bonds:	#⟨BOND b-T0775⟩
connecting-atoms:	#⟨ATOM C-T0768⟩
group-weight:	15.035001
parent-molecule:	#⟨ORGANIC-MOLECULE C2H6-T0779⟩
DB-group:	#⟨DB-GROUP methyl⟩

arranged in increasing strength. This is shown below:

IDENTIFY-WEAKEST-BOND applied to `bond-abstraction` ⇒
 ((`bond-1`) (`bond-2 bond-3 bond-4 bond-5 bond-6 bond-7`))

where

`bond-1` = carbon-carbon single bond connecting `methyl-1` to `methyl-2`
`bond-2` to `bond-4` = carbon-hydrogen single bond on `methyl-1`
`bond-3` to `bond-7` = carbon-hydrogen single bond on `methyl-2`

Bond selection is based on relative strength. A default energy difference of 10 kcal/ mol is used by IDENTIFY-WEAKEST-BOND, but this value can be changed by the user. Bonds of equivalent strength are listed together.

For each system of reactants, IDENTIFY-WEAKEST-BOND identifies the weakest substrate bond, and compares it with others in the system to ensure minimal global bond strength. Selected instances of bond are then appended to potentially-cleavable-bonds.

The weakest system `bond` is identified by applying the method IDENTIFY-ABSOLUTE-WEAKEST to the bond representation of `ethane`. The value of **weakest-bond** is the carbon–carbon single bond connecting `methyl-1` to `methyl-2`. Specifically

IDENTIFY-ABSOLUTE-WEAKEST-BOND
applied to potentially-cleavable-bonds `bond-1`

Once selected, it is associated with the global queue **weakest-bond**. Specifically

weakest-bond `bond-1`

This queue prevents $K_{\mathtt{initiation}}$ from processing the weakest bond of individual molecules when the energy difference between those bonds is greater than the default value. It also ensures that the initiation process is focused on substrates that are most likely to cleave in `reaction-environment`. For example, in a complex molecule or set of species where one particular bond is substantially weaker than others in the system (e.g., a peroxy bond), knowledge of **weakest-bond** focuses the initiation process on that bond. Once potentially-cleavable-bonds and **weakest-bond** become known to $K_{\mathtt{initiation}}$, the method, FEASIBLE-P evaluates the energy in `reaction-environment` and determines whether there is sufficient energy for bond cleavage. The method SUFFICIENT-THERMAL-ENERGY-P performs this evaluation. When FEASIBLE-P evaluates true, $K_{\mathtt{initiation}}$ calls K_t on the bond selected for cleavage. Application of bond cleavage operations, operators of $\kappa_{\text{ab-initio}}$, to the weakest bond of `ethane` produces a `microhomolysis-reaction`. Each `micro-reaction` has associated with it reactants, products, and reaction stoichiometry. In addition, the bond selected for cleavage is explicit, as are various other properties of the `reaction`. Methods of kinetic operator are used to identify these properties; they include COLLECT-PRODUCTS, COMPUTE-STOICHIOMETRY, COMPUTE-BOND-FAVORABILITY, ASSESS-THERMODYNAMIC-FAVORABILITY, and COMPUTE-EQUILIBRIUM-CONVERSION.

Information pertinent to the transformation is associated with `micro-reaction` to make it easily accessible by methods and operations external to `micro-reaction`. Information essential to the external environment includes:

```
micro-homolysis-reaction
    reactants:       (ethane)
    products:        (methyl-radical-1  methyl-radical-2)
    stoichiometry:   ((ethane.-1)
                     (methyl-radical-1.+1)
                     (methyl-radical-2.+1))
    bond-cleaved:    bond-1
    environment:     reaction-environment
```

Since multiple bonds may reside in potentially-cleavable-bonds, each possibly leading to a `micro-reaction`, $\kappa_{\text{initiation}}$ creates a `global-reaction` that acts as a place holder for managing high-level information: bond-queue and `micro-reactions`. `Global-reactions` maintain a record of this information and the association of a `micro-reaction` corresponding to a particular bond cleavage:

```
global-homolysis-reaction
    bonds-to-be-cleaved: (bond-1...bond-n)
    weakest-bond: (bond-1)
    micro-reaction: (micro-homolysis-1...micro-homolysis-n)
```

K_t, the function responsible for performing a particular transformation, calls methods contained in $\kappa_{\text{ab-initio}}$. Principal methods invoked by K_t to effect homolytic bond cleavage of `ethane` are GENERAL-SCISSION and CLEAVE-BOND. The methods often utilize helping functions, contained in various modeling elements (e.g., IDENTIFY-BOND-TYPE, HOMOLYTIC-P, IDENTIFY-CHARGE-DISTRIBUTION), to assist the transformation process. After a molecular bond has been selected for cleavage, it is passed to GENERAL-SCISSION by $\kappa_{\text{ab-initio}}$. Operations composing GENERAL-SCISSION identify the bonds parent-structure, perform the cleaving function, and manage fragments resulting from the cleavage process.

Fragment management is facilitated by ABSTRACT-GROUPING and ABSTRACT-ATOM-GROUPING. These methods partition in abc, given a set of starting points (e.g., atoms). Normally, these points are specified by the endpoints (e.g., connecting atoms) of the reaction center identified by $K_{\text{get-sites}}$. In ethane disassociation, the reaction center endpoints are the two carbon atoms associated with the carbon–carbon single bond. As a consequence, in the application of CLEAVE-BOND to weakest-bond (i.e., the element specified by **weakest bond**), two abc's are identified. These are grouped by ABSTRACT-ATOM-GROUPING and become radicals (i.e., methyl-radical-1 and methyl-radical-2 on instantiation by abc).

The (abc) instantiation process evaluates, updates, and classifies the abc as an instance of radical. During instantiation, abc invokes MAP-OLD-ATOM-TO-NEW-ATOM, a method that maintains pointers within the chemical system so that atoms constituting a substrate know where they came from. This is accomplished using attribute's identifier and old-identifier. During the dissociation of ethane the value of carbon-1 identifier is "C-T0764," whereas its old-identifier has value nil, reflecting external creation for ethane (i.e., database for user specification of the abc). However, the attribute value of old-identifier for carbon-3, the carbon making up new methyl radical, methyl-radical-1, reflects its progenitor carbon-1:

carbon-3		carbon-1	
identifier:	"C-T0790	identifier:	"C-T0764
old-identifier:	"C-T0764"	old-identifier:	nil
. . .	. . .	. . .	. . .

Since the value of old-identifier describing carbon-3 has the same value as carbon-1's identifier value, these pointers enable construction of a substrate's complete history together with the operators that enabled each transformation.

A history trace is constructed by chaining through the values of abc attribute progenitors. Similarly, the values of enabling-conditions, an attribute describing reaction, is traced to identify the various reactions constituting a particular pathway. These utilities, in combination, allow every attribute describing a modeling element (e.g., abc) to be logged and accessed, in chronological order. This schema is one way in which LCR affords multifaceted and multilevel description of a modeling element throughout its history, and allows, for example, a multilevel description of a particular chemical species and the reaction pathways in which it participates throughout its life cycle.

Once a composite operation has successfully been executed, FEASIBILITY-P returns true and ab initio operations making up the sequence of transformations contained in K_t return viable transformations; unique products identified by `reaction` are appended to the global queue denoted **potential-reactant-queue**. This queue contains a complete listing of potential reactants throughout the reaction cycle. Management of potential reactants in this manner allows the behavior of a system as it moves toward an equilibrium state, to be simulated.

Continued application of composite operations, invoked by FIND-ALL-PATHWAYS on the substrates contained in **potential-reaction-queue** identifies additional instances of `transfer, propagation,` and `termination reaction.` By tracing the history of these instances of `reaction, free-radical-reaction` (i.e., free-radical pathways), one can construct a free-radical reaction. Methods of `free-radical-reaction` provide the utility for searching through a set of chemical species, using the value of the progenitors attribute and MOLECULE-EQUIVALENT-P, to identify substrate loops within the propagation reaction network. Propagation reactions are then identified and the attribute value established.

However, with the myriad products capable of being formed in termination sequences, application of $K_{\mathtt{free\text{-}radical}}$ to substrates in the **potential-reactant-queue** may not terminate. This potential exists because new reactants, resulting from `coupling-reactions` and `combination-reactions`, are constantly appended to **potential-reactant-queue**. For example, two `ethyl-radicals` can combine to form `butane`, and hydrogen abstraction of `butane` can form `butyl`, which may combine to form `octane`. This cycle can, in principle, continue indefinitely. To prevent such cycles, substrates are removed from **potential-reactant-queue** when they have been consumed, or when a homologous series of a reactant has been previously examined. The latter is accomplished using HOMOLOGOUS-SERIES-P, with the register function responsible for appending potential reactants to **potential-reactant-queue**.

Pathways of the overall reaction sequence are constructed using instances of `free-radical-reaction`. Information associated with the individual instances `initiation-reaction, transfer-reaction, propagation-reaction, branching-reaction,` and `termination-reaction` facilitates this task. With this information, `free-radical-reaction` is able to construct individual free radical pathways.

IV. MODEL.LA.: A Modeling Language for Process Engineering

MODEL.LA. is a high-level, special-purpose language, which was developed to support the modeling activities of a broad range of process engineering tasks; process design, simulation, operations planning, diagnosis, configuration of control systems, and others. In this regard, MODEL.LA. is quite distinct in both scope and capabilities from other modeling languages such as ASCEND, MODASS, and OMOLA. The technical details of MODEL.LA.'s structure and implementation can be found in Stephanopoulos *et al.* (1990a, b) and Henning and Leone (1990).

MODEL.LA. shares many common features with LCR. Both languages share a common set of semantic relationships and a common syntax. They differ in their modeling elements, which reflect the different vocabularies of chemistry and process engineering. Nevertheless, the modeling elements of each language are organized into subsets, which depict the *structural* and *behavioral* knowledge of the corresponding domains. In this section we will provide an overview of MODEL.LA.'s structure, and in Section V we will discuss the utilization of MODEL.LA. within the scope of engineering problems.

A. Basic Modeling Elements

To account for all representational needs in process engineering, MODEL.LA uses six (6) basic classes of modeling elements, and fairly rich modeling hierarchies of subclasses, emanating from the basic modeling elements.

1. *Elements Defining Structures*

To represent topological structures of processing systems, MODEL.LA employs the following modeling elements:

a. Modeling-Element 1: `generic-unit`. This is used to capture a system at any level of detail. Thus, an overall plant, a plant-section, a processing unit, a part of a processing unit (e.g., tubes of a heat exchanger), a phase in a processing vessel (e.g., liquid 1 in a two-liquid phase reactor), are all represented as instances of `generic-unit` (or, its specialized subclasses, as we will see in subsequent section). Also, sensors, actuators, control loops, and information processing systems are all represented and instances of `generic-unit`. Each instance of a generic unit encapsulates its own boundary, internal structural components, modeling relationships, and associated assumptions.

b. Modeling-Element 2: `port`. These are special objects that the generic units use to pass information to each other. Thus, the boundary of a generic unit is in essence defined by the set of ports through which it communicates with the rest of the world. Flow of materials, energy, or information into or from a generic unit takes place *only* through a port.

c. Modeling-Element 3: `stream`. These objects relate the ports of connected generic units, and are the conduits through which material, energy, information, or other quantities pass from one generic unit to the next. The type of a specific stream is always the same as the type of the ports that is associated with it.

2. Elements Defining Behavior

The following three modeling elements are used to capture the behavioral characteristics of materials and processing systems.

a. Modeling-Element 4: `constraint`. This class is used to capture any piece of knowledge associated with a declarative relationship among variables and parameters, such as a logical, qualitative, or quantitative relationship. An instance of `constraint` (or, its subsidiary subclasses) contains information about the form of the relationship it represents, the terms and variables that compose it, the "meaning" and significance of the relationship, as well as the range of its applicability.

b. Modeling-Element 5: `generic-variable`. Instances of this class (or its subsidiary subclasses) constitute the building blocks for the construction of modeling relationships. They represent parameters and variables describing physical quantities, engineering terms, design or operating specs, etc. An instance of `generic-variable` encapsulates information about the physical significance, value, range of possible values, trends over time, units, and other parameters of the quantity it represents.

c. Modeling-Element 6: `modeling-scope`. This object is a list of consistent instances of `constraint`, which represent the declarative relationships that apply to all components of a model, i.e., assumptions, simplifications, conjectures expected to be true, and modeling relationships. It is clear that `modeling-scope` is a *redundant* modeling element (e.g., all information in an instance of `modeling-scope` is contained in a set of instances of `constraint`). The importance, though, of maintaining a high-level container of the context in which the model was developed is sufficient to warrant the element's elevation into a basic element of the modeling language, allowing easier and direct manipulation of the modeling context.

B. Semantic Relationships

MODEL.LA. possesses the same set of 13 semantic relationships as LCR, with the same semantic implications:

1. *Semantic-Relationship 1:* ***is-attribute-of.*** Indicates that an entity is an attribute of a particular class of objects.

   "entity" ***is-attribute-of*** `object-class`

2. *Semantic-Relationship 2:* ***is-method-of.*** Indicates that a particular algorithmic procedure is a method of a specific class of objects.

   PROCEDURE ***is-method-of*** `object-class`

3. *Semantic-Relationship 3:* ***is-a.*** Relates a subclass to the mother class:

   object-subclass ***is-a*** object-class

4. *Semantic-Relationship 4:* ***is-a-member-of.*** Relates an instance to the class that generated it:

   instance-of-class-*x* ***is-a-member-of*** class-*x*

5. *Semantic-Relationship 5:* ***is-composed-of.*** Relates a modeling object to its component parts:

   jacketed-CSTR ***is-composed-of*** (jacket; stirred-tank)

6. *Semantic-Relationship 6:* ***:is-part-of.*** It is the semantic relationship symmetrical to the previous one.
7. *Semantic-Relationship 7:* ***is-attached-to.*** This relationship is used primarily to define the boundary of a generic unit, by defining the ports

attached to a specific unit:

`part-x` ***is-attached-to*** `unit-y`

8. *Semantic-Relationship 8:* ***is-connected-by.*** It is the inverse of the previous one.
9. *Semantic-Relationship 9:* ***is-described-by.*** It is used to indicate the variables or relationships used to describe the behavior of a generic unit:

   unit-*x* ***is-described-by*** (variable-*y*; equation-*z*; rule-*w*)

10. *Semantic-Relationship 10:* ***is-describing.*** This is the inverse of the previous one.
11. *Semantic-Relationship 11:* ***is-characterized-as.*** It is used to specialize a modeling class by specifying a fixed, default value for a given attribute of the class; for example, the following statement

    `Reactor-x` ***is-characterized-as*** "isobaric"

    puts a constant value in the attribute, "pressure", of the class `Reactor-x`. All instances of reactor-*x* will have constant pressure.
12. *Semantic-Relationship 12:* ***is-disaggregated-in.*** As in LCR, this semantic relationship associates a modeling element, located in a given context, with its constituent modeling components that are located at a different (more detailed) context. For example,

    plant-*x* ***is-disaggregated-in*** (reaction-section; separation-section)

13. *Semantic-Relationship 13:* ***is-abstracted-by.*** This is the inverse of the previous one.

The semanic relationships of MODEL.LA. obey the same axioms as those of LCR, namely *transitivity*, *monotonicity*, *commutativity*, and *merging*. For more details, see Section III.D.

C. Hierarchies of Modeling Subclasses

Six basic modeling classes presented in Section IV.A are the root-classes for expanded trees of modeling subclasses. Whereas the six basic elements are designed to possess generic attributes and methods, independent of the particular domain they represent, the hierarchical trees emanating from

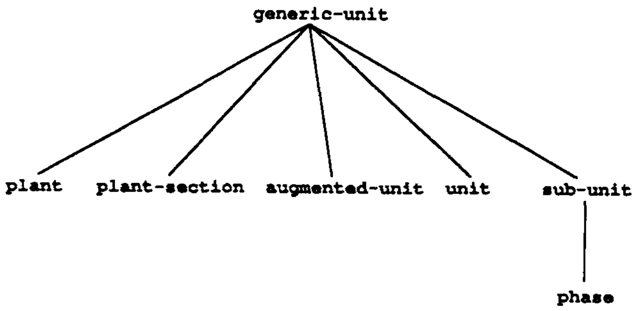


FIG. 17. The tree of `generic-unit` modeling subclasses.

them include modeling subclasses that are specialized to *reflect the vocabulary of process systems engineering*:

1. *The tree of* `generic-unit` *classes* (Fig. 17). The subclass `generic-processing-unit`, encompasses the various abstractions of processing systems: `plant`, `plant-section`, `augmented-unit`, `sub-unit`. The `sub-unit` has a specific subclass, `phase`, to capture the specialized attributes of a material with uniform thermodynamic state.

2. *The tree of* `port` *classes*. Four subclasses of ports emanate from the basic modeling element; `convective-port`, `material-port`, `energy-port`, and `information-port`. Each is specialized to express the characteristics of the quantity that flows through it.

3. *The tree of* `stream` *classes*. The number of stream subclasses is equal throughout and reflect similar structure of attributes as the `port` subclasses.

4. *The tree of* `modeling-scope` *classes*. It has two subclasses: `model` and `context`. The first captures all those relationships that reflect assumptions, hypotheses, and conjectures, all of which define the contextural scope of a model. The second captures the list of mathematical relationships describing the behavior of a generic-unit.

5. *The tree of* `constraint` *classes*. (Fig. 18). The `assignment` subclass is used to represent constraints, which are *solved* with respect to a particular variable. The `relationship` subclass is used to represent *unsolved* constraints.

6. *The tree of* `generic-variable` *classes*. This is composed of two subclasses, the `variable` and the `term`. The most important subclass is the `term`, which represents a compound quantity, made up from the

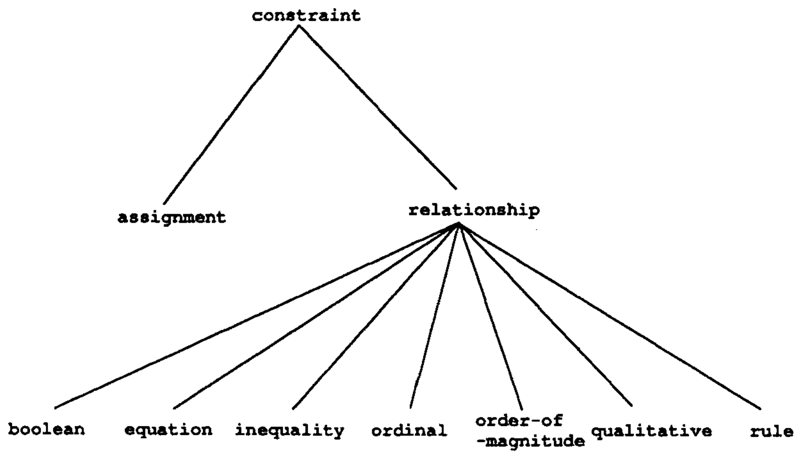


FIG. 18. The tree of constraint modeling subclasses.

combination of other elementary variables and parameters. It is used to model; terms appearing in modeling equations (e.g., enthalpy of stream-*i*, accumulation), dimensionless numbers [e.g., Re, Pr, Nu (Reynolds, Prandtl, Nusselt numbers)], design quantities [e.g., L/mG (liters per milligauss) of absorbers].

D. SYNTAX

MODEL.LA., like LCR, uses the BNF grammar rules for the construction of syntactically correct modeling sentences. For more details on the syntactic characterization of new models, the reader is referred to Stephanopoulos *et al.* (1990a, b), and MODEL.LA.'s manual (Henning and Leone, 1990).

V. Phenomena-Based Modeling of Processing Systems

MODEL.LA. is a language with infinite extensibility of its vocabulary, enabled by a fixed set of six modeling hierarchies and a fixed set of 13 semantic relationships. In Section IV.C, we discussed the hierarchies of modeling subclasses emanating from the six basic modeling elements. What is far more important for the modeling power of MODEL.LA. is its ability to capture all aspects of chemical engineering science, using the physical and chemical phenomena as the basis for the automatic generation of models for processing systems, using as input the declared physical and chemical characteristics of a processing system. For example, the user (or, another program) declares the following about a reactor-*x*:

Reactor-x is a two-liquid-phase jacketed CSTR.

MODEL.LA. is capable of parsing the sentence and of automatically creating the following "interpretation":

1. Reactor-*x* is composed of two subsystems: jacket and continuous stirred-tank.
2. The reacting mixture in the stirred-tank is made up of two liquid phases.
3. There is mass transfer between the two liquid phases.
4. There is heat transfer between reacting mixture and jacket.
5. Materials enter and leave the reactor through convective flows.
6. Enthalpy enters and leaves the reactor through convective flows.

Once this "interpretation" has been established, MODEL.LA. (a) generates all the requisite modeling elements and (b) constructs the modeling relationships, such as material balances, energy balance, heat transfer between jacket and reactive mixture, mass transport between the two liquid phases, equilibrium relationships between the two phases, estimation of chemical reaction rate, estimation of chemical equilibrium conditions, estimation of heat generated (or consumed) by the reaction, and estimation of enthalpies of material convective flows. In order to automate the above tasks, MODEL.LA. must possess the following capabilities:

1. Rich hierarchies of modeling elements, which can be used to represent any conceivable quantity of interest in chemical engineering science.
2. A series of procedures, which can automatically generate the complete set of modeling equations representing the behavior of a processing system.

A. The "Chemical Engineering Science" Hierarchies of Modeling Elements

The modeling hierarchies of `constraint` and `generic-variable` (see Section IV.C) have been expanded to a series of subclasses, which have been specialized to capture the knowledge of chemical engineering science, and use it in an explicit manner during the construction of models for processing systems.

1. Classes of Variables

Class `variable` is a subclass of the basic class `generic-variable.` From the class `variable` emanates the tree of subclasses, a partial view of which is shown in Fig. 19. Unlike other modeling approaches, MODEL.LA. does not represent variables through their values alone, but it provides an extensive structure that includes many additional attributes in the description of a variable. The additional attributes allow MODEL.LA. to reason about these variables and not just acquire their values. Thus, a set of methods in the class `variable` allow any of the subclasses to monitor their values, react with predefined procedures when the value of the variable changes, invoke values from external databases, and so on.

2. Classes of Terms

The class `term` is a subclass of the `generic-variable`, and one of the most important modeling elements in MODEL.LA. It is used to represent a very broad spectrum of compound physical quantities, which appear in modeling relationships. Figure 20 shows a partial view of that

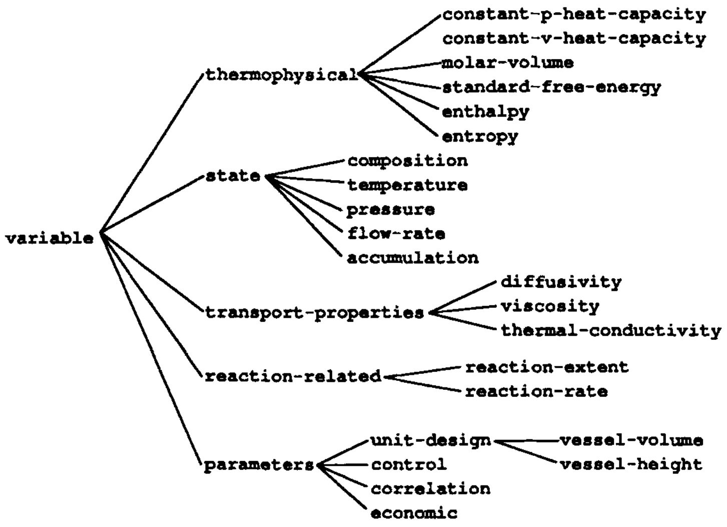


FIG. 19. Partial view of the `variable` hierarchy of modeling subclasses.

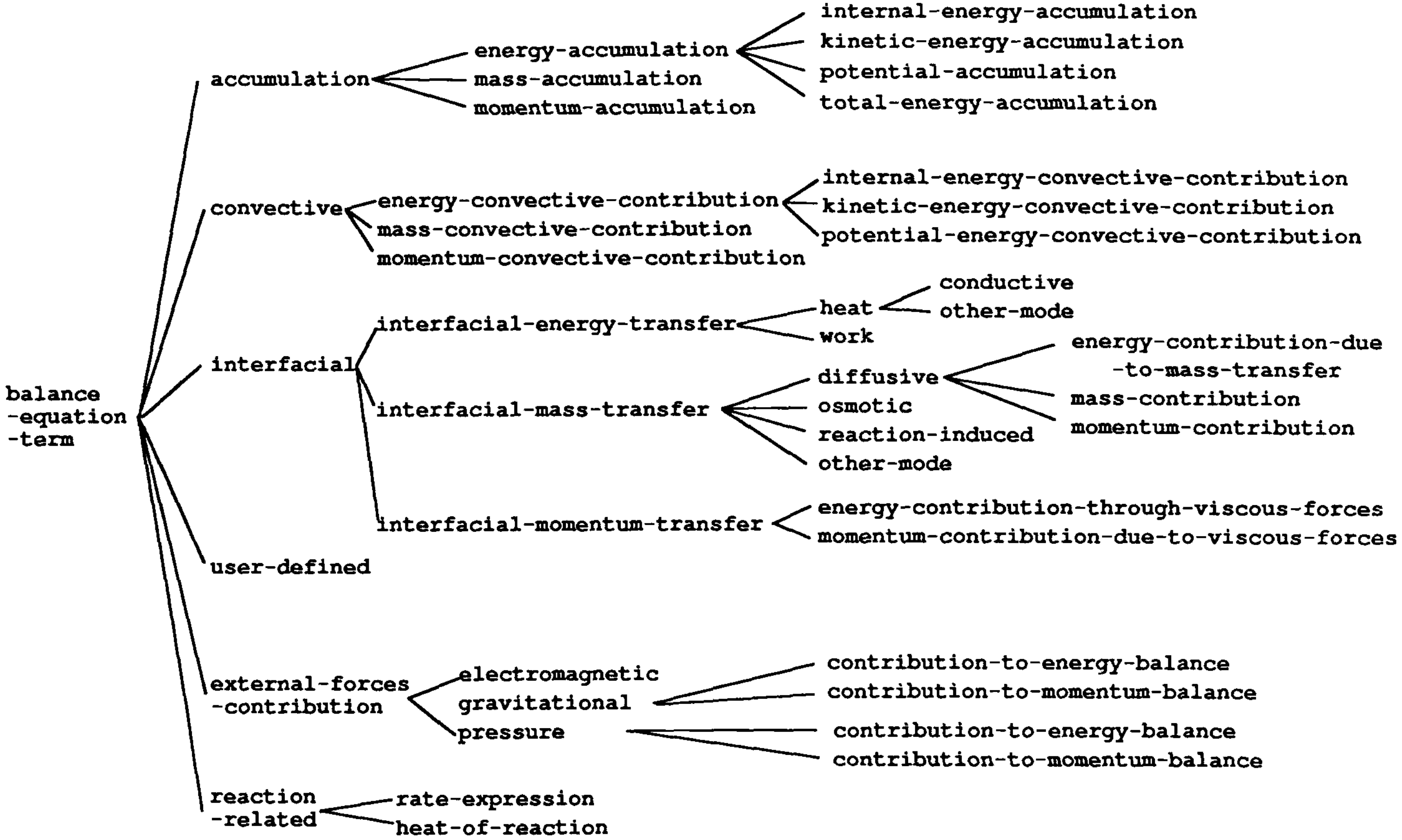

balance -equation -term
accumulation
energy-accumulation
mass-accumulation
momentum-accumulation
internal-energy-accumulation
kinetic-energy-accumulation
potential-accumulation
total-energy-accumulation
convective
energy-convective-contribution
mass-convective-contribution
momentum-convective-contribution
internal-energy-convective-contribution
kinetic-energy-convective-contribution
potential-energy-convective-contribution
interfacial
interfacial-energy-transfer
heat
conductive
other-mode
work
interfacial-mass-transfer
diffusive
energy-contribution-due -to-mass-transfer
mass-contribution
momentum-contribution
osmotic
reaction-induced
other-mode
interfacial-momentum-transfer
energy-contribution-through-viscous-forces
momentum-contribution-due-to-viscous-forces
user-defined
external-forces -contribution
electromagnetic
gravitational
contribution-to-energy-balance
contribution-to-momentum-balance
pressure
contribution-to-energy-balance
contribution-to-momentum-balance
reaction -related
rate-expression
heat-of-reaction

FIG. 20. Partial view of the modeling hierarchy of `term` class, which represent terms in balance equations.

part of the hierarchy of classes emanating from `term`, which are involved only in the definition of balance equations during the modeling of processing units. The complete structure of the hierarchy of terms is fairly extensive and covers the bulk of terms appearing in balance equations, transport phenomena, and equilibrium relationships.

3. Classes of Equations

All these modeling elements emanate from the class `equation` (Fig. 18), and a partial view is shown in Fig. 21. Each subclass captures all the information about a given equation, including its significance, preconditions for its correct use, and implications on the physics of the modeled system.

B. Formal Construction of Models

Every model constructed through MODEL.LA. is represented through a MCDD (see Section III.B), with the modeling elements playing the role of nodes and the semantic relations among the modeling elements representing the edges of the digraph. Figure 22 shows the MCDD associated with the model of a continuous-stirred-tank reactor (CSTR) without a jacket. Note the nodes representing topological elements (e.g., the ports, `input1-vessel` and `input2-vessel`), variables (e.g., `pressure, temperature, composition`), modeling equations (e.g., `mass-lumped-balance, energy-lumped-balance, phase-equilibrium-equation`). The element `vessel-surroundings-heat-transfer` is not specified. If it is specified to be a jacket, then the MCDD of the jacket is appended to that of the CSTR and produces the composite MCDD of Fig. 23, which represents the model of a *jacketed CSTR*. This type of modular construction of processing models provides MODEL.LA. with infinite flexibility in representing any processing systems. Furthermore, predefined models can be used as components in future representations, making the next modeling effort no more difficult than previous ones.

C. Multifaceted Modeling of Processing Systems

Consider the three abstractions of the plant shown in Fig. 1. How could one use the capabilities of MODEL.LA. to generate consistent represen-

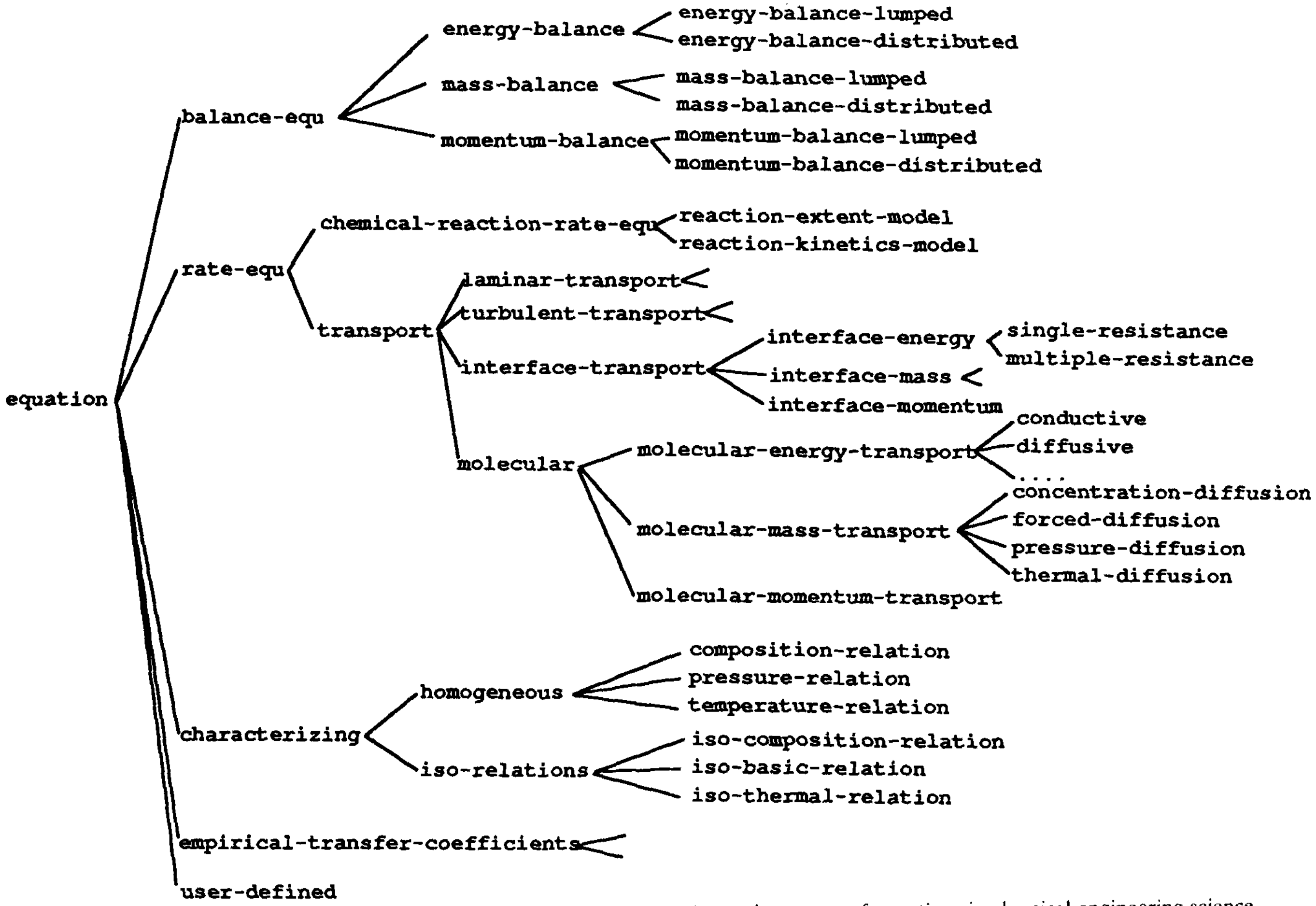


FIG. 21. Partial view of the hierarchy of modeling classes representing various types of equations in chemical engineering science.

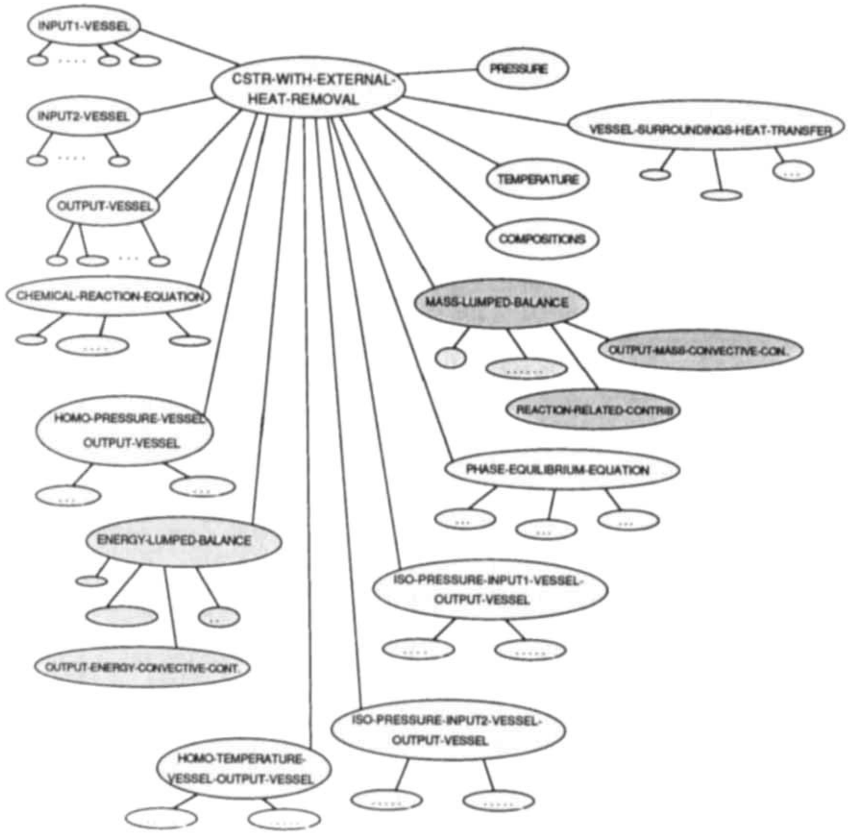

INPUT1-VESSEL
CSTR-WITH-EXTERNAL-HEAT-REMOVAL
PRESSURE
INPUT2-VESSEL
VESSEL-SURROUNDINGS-HEAT-TRANSFER
TEMPERATURE
OUTPUT-VESSEL
COMPOSITIONS
CHEMICAL-REACTION-EQUATION
MASS-LUMPED-BALANCE
OUTPUT-MASS-CONVECTIVE-CON
REACTION-RELATED-CONTRIB
HOMO-PRESSURE-VESSEL-OUTPUT-VESSEL
PHASE-EQUILIBRIUM-EQUATION
ENERGY-LUMPED-BALANCE
ISO-PRESSURE-INPUT1-VESSEL-OUTPUT-VESSEL
OUTPUT-ENERGY-CONVECTIVE-CONT
ISO-PRESSURE-INPUT2-VESSEL-OUTPUT-VESSEL
HOMO-TEMPERATURE-VESSEL-OUTPUT-VESSEL

FIG. 22. The MCDD associated with the model of a CSTR without jacket. (Reprinted from *Comp. Chem. Eng.* **14**, Stephanopoulos, G., Henning, G., and Leone, H. MODEL. LA A modeling language for process engineering. Part I, Page 813, Copyright 1990, with kind permission from Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

tations of the three views of the same plant? On the other hand, the two versions of the plant shown in Fig. 2 have many modeling elements in common. How could one use MODEL.LA. to maintain two distinct facets of the same plant at the same level of detail? Such multiviewing of process models, along with the corresponding multilevel and multicontext requirements, define the scope of the so-called *multifaceted modeling of processing systems*.

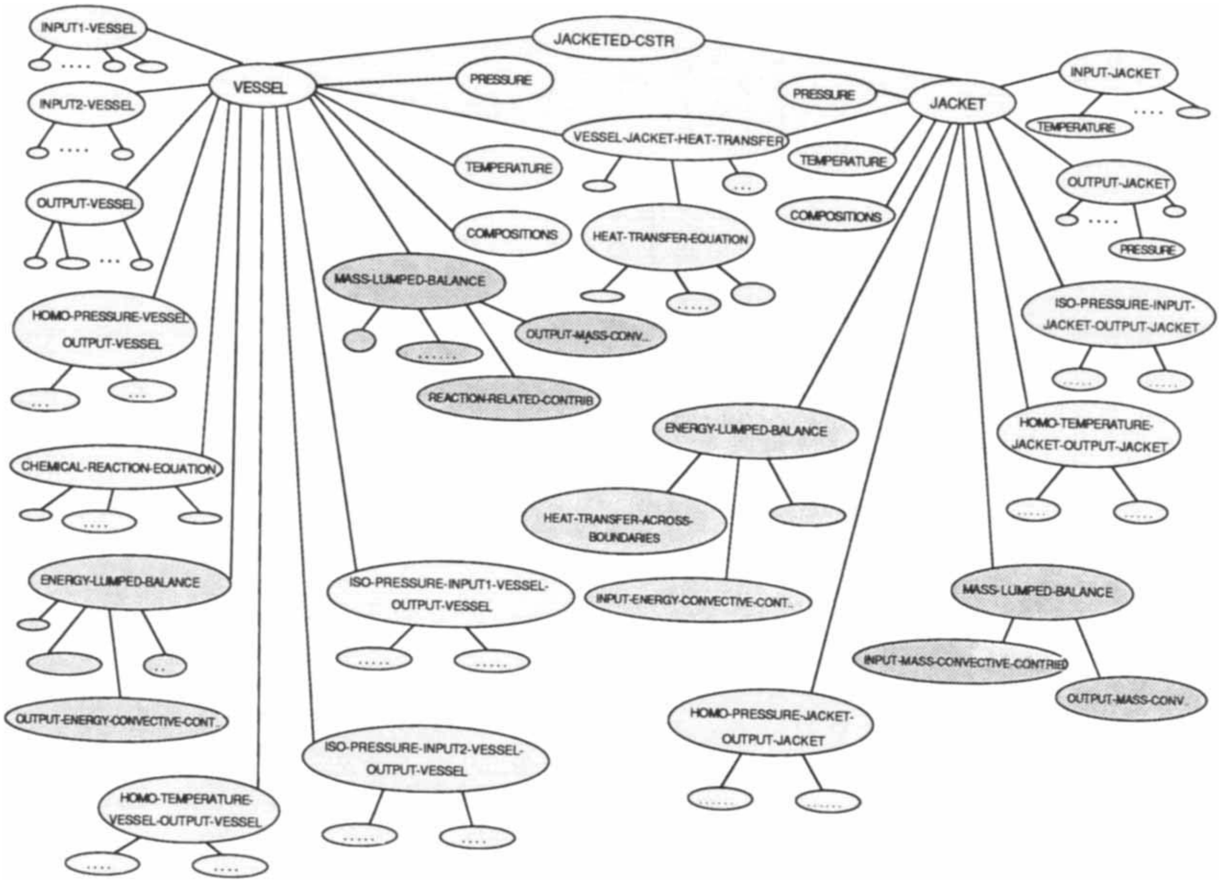

JACKETED-CSTR
VESSEL
JACKET
INPUT1-VESSEL
INPUT2-VESSEL
OUTPUT-VESSEL
HOMO-PRESSURE-VESSEL OUTPUT-VESSEL
CHEMICAL-REACTION-EQUATION
ENERGY-LUMPED-BALANCE
OUTPUT-ENERGY-CONVECTIVE-CONT.
HOMO-TEMPERATURE-VESSEL-OUTPUT-VESSEL
PRESSURE
TEMPERATURE
COMPOSITIONS
MASS-LUMPED-BALANCE
OUTPUT-MASS-CONV.
REACTION-RELATED-CONTRIB
ISO-PRESSURE-INPUT1-VESSEL-OUTPUT-VESSEL
ISO-PRESSURE-INPUT2-VESSEL-OUTPUT-VESSEL
VESSEL-JACKET-HEAT-TRANSFER
HEAT-TRANSFER-EQUATION
ENERGY-LUMPED-BALANCE
HEAT-TRANSFER-ACROSS-BOUNDARIES
INPUT-ENERGY-CONVECTIVE-CONT.
HOMO-PRESSURE-JACKET-OUTPUT-JACKET
INPUT-JACKET
OUTPUT-JACKET
ISO-PRESSURE-INPUT-JACKET-OUTPUT-JACKET
HOMO-TEMPERATURE-JACKET-OUTPUT-JACKET
MASS-LUMPED-BALANCE
INPUT-MASS-CONVECTIVE-CONTRIB
OUTPUT-MASS-CONV.

FIG. 23. The MCDD associated with a jacketed CSTR. (Reprinted from *Comp. Chem. Eng.* **14**, Stephanopoulos, G., Henning, G., and Leone, H. MODEL. LA A modeling language for process engineering. Part I, Page 813, Copyright 1990, with kind permission from Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

1. Concurrent Models at Multiple Abstractions

Consider again the "overall process" of Fig. 1a. It can be represented by an instance of the modeling element, `generic-unit` (or, `plant`, if we want to take advantage of its special attributes). MODEL.LA. can derive automatically the necessary modeling instances for the refined plant of Fig. 1b, through the invocation of the following semantic relationship:

overall-process ***is-disaggregated-in*** (generalized-reaction-section; generalized-separation-section)

If the `generalized-reaction-section` and `generalized-separation-section` are defined as instances of the class `plant-section`, then the correspondence between the various ports in Fig. 1a and 1b is automatically established. Thus, all the "external" ports `port-A`, `port-B`, `port-P`, and `port-BP` in Fig. 1b acquire the same information that their "external" counterparts possess in Fig. 1a. Furthermore, MODEL.LA. automatically establishes instances of the streams `Gas-Recycle` and `Liquid-Recycle` and associates them with the corresponding ports of the two plant sections in Fig. 1b. The automatic generation of models for the refined representation of the same plant as in Fig. 1b is carried out by a series of procedures that MODEL.LA. activates when an instance of the `generic-unit` (or its subclasses) is (are) refined through the ***is-disaggregated in*** semantic. Specifically, these procedures carry out automatically the following tasks:

1. *Establish structural compatibility* between the abstract and the set of refined generic units.
2. *Establish topological compatibility* between the ports and streams of the abstract entity and those of the refined entities generated after the disaggregation process.
3. *Establish compatibility between the physicochemical phenomena* occurring at the abstract and refined contexts of the plant.
4. *Propagate values of variables* from the abstract to the more detailed context (or, vice versa), using the compatibility links established above.

For the technical details of the procedures, carrying out the above tasks, the reader is referred to Stephanopoulos *et al.* (1990a, b).

2. Concurrent Models at the Same Abstraction

MODEL.LA. maintains concurrent models of the same process at the same level of abstraction using the notion of *context* (see also Section

III.D). For example, let CONTEXT-1 signify the set of (1) topological relationships characterizing the structure in Fig. 2a and (2) modeling relationships describing the functionality and behavior of the generalized reaction and separation sections. Then, we can create CONTEXT-2 as the child of CONTEXT-1, by changing a few of its relationships so that the new context can capture the "splitter" and the "purge-stream" of Fig. 2b. Since CONTEXT-2 (1) inherits all relationships from CONTEXT-1, (2) overrides or eliminates those that changed, and (3) adds new modeling elements and relationships, the two contexts share the common modeling elements (without duplication) and maintain their distinctive character through their private modeling elements or/ and relationships. For more technical details on the MAT, see Section III.D; and for the knowledge sharing of concurrent models at the same level of abstraction, see Stephanopoulos *et al.* (1990a, b).

D. Computer-Aided Implementation of MODEL.LA.

MODEL.LA. was implemented within IntelliCorp's KEE, using Common LISP (list processing language) as the low-level programming language. KEE's frame system provided a fairly flexible, object-oriented programming environment. To take advantage of a number of facilities and to be integrated with an advanced, computer-aided engineering framework, MODEL.LA. was incorporated into the DESIGN-KIT (Stephanopoulos *et al.*, 1987).

1. Organization

The different objects comprising the modeling system are grouped into several knowledge bases:

- The basic modeling elements of the language, and the subclasses that emanate from them, are stored in a knowledge base (KB) called NETWORK-MODEL.
- When the composite objects representing new model classes are generated, their parts constituent are created as more specialized subclasses of the modeling elements of the NETWORK-MODEL KB. However, all the specialized entities constituting model classes are stored in the LIBRARY KB.
- When a model class is instantiated to represent a given model situation, all the entities used to generate the composite object that represents the model instance are created in the WORKING KB.

The rationale behind this division is that managing large knowledge bases is difficult. As knowledge bases grow in size and complexity, they strain the capacities of software tools for knowledge editing and maintenance. This division is based on the different functionality of the several KBs. The NETWORK-MODEL KB contains the basic knowledge, and consequently, is almost fixed. It will only be changed if the language is upgraded or modified. The LIBRARY KB will change if the model classes are modified or new ones are defined. Finally, the WORKING KB contains the most volatile type of knowledge, that is, entities that are created to represent very specific situations.

The WORKING KB stores all the instance objects that are used to represent the models employed during a particular process engineering activity. As was shown in previous sections, however, a model may have multiple descriptions of its structure, topology, behavior, etc. These different modeling situations are encapsulated in CONTEXTS. The CONTEXT facility is built on top of KEE worlds. KEE worlds is a set of tools provided by the KEE system to allow the representation of alternative states of the knowledge base, and to facilitate the exploration and comparison of alternative scenarios. The information in a standard KEE knowledge base provides a starting point for modeling hypothetical situations represented as worlds. In our case this information, called the "background," is stored in the WORKING KB. This KB contains the assumptions that are true in every hypothetical situation, and perhaps the problem-specific facts describing the initial modeling situation.

2. *Facilities for the Generation of Models*

Model classes can be generated in one of two ways:

(a) By using the MODEL.LA. language, a natural language built on top of KEE's Tell And Ask. The Tell And Ask language is a special facility for interacting with a knowledge base, by telling it new facts to incorporate, and asking about the facts it already contains. It is basically a set of KEE functions and a set of linguistic forms that format information for KEE.

(b) By using the MODEL-EDITOR, that is a specially designed mouse-and-menu-style interface. The description of the MODEL-EDITOR interface can be separated clearly in two parts: one governing the appearance and properties of the display itself, and the other defining the logical relations between display and modeling objects.

Our application domain makes it natural to adopt menus as the underlying display metaphor. *Menus* are represented as structured objects organized in *panels*. Menus themselves are constructed of rectangular primitive regions called *boxes* and nonprimitives called *polyboxes*. Boxes (which need not have visible outlines) are used to accept input data (typed or mouse-selected data), to allocate regions for displaying text and annotations of various kinds, etc. The adoption of a menu-driven interface has several advantages, among which we note the following: (1) *error prevention*, which is achieved naturally, because menus give the user the complete list of assumptions that can be specified at any point during the definition of a model; (2) *memorization*, to avoid or minimize the need for reference to an external manual or calls to help facilities; (3) *immediate visual feedback*; and (4) *active menus*, which help ensure that all the assumptions that need to be set for the model to be defined properly are in fact set. The model editor interface has been designed according to a "conditional display" paradigm. The complete definition of a model class can in principle require the user to specify a very large number of individual assumptions. In practice, fortunately, most model classes can be defined supplying only a small set of the possibly relevant data.

3. *Facilities for User-MODEL.LA. Interaction*

Specially designed tools provide an integrated set of knowledge access facilities serving various purposes such as inspecting the current status of the several KBs, linking the available model classes and their associated assumptions, displaying the structure of models or just partial views of them, listing the assumptions associated with a model instance in several contexts where it is defined, listing the relations and variables associated with a model instance, and listing the variable values.

To the extent that the structure of a model reflects the logical relations among its several constituent entities, models can be verified in a straightforward way, by confirming that the necessary links exist. Thus, specific browsers are used to display the different specification semantic relations existing among the pieces of a model. It is obvious that this browser facility allows a selective access to the different views of a model.

Other types of browsers are employed to show different semantic relations between the components of the language, for example, to display "***is-a***" or "***is-a-member-of***" links, or to show the directed graph that is formed by the contexts and the parent/ child relations between them.

Summarizing, we can say that the user-interaction facilities incorporated in this system are governed by the need to provide a smooth, natural, and efficient environment for accessing, analyzing, and debugging knowledge.

References

Andersson, M., An object-oriented modeling environment. *In* "Simulation Methodologies, Languages and Architectures and AI and Graphics for Simulation" (Iazeolla *et al.*, eds.), 1989 European Simulation Multiconference, Rome, pp. 77–82. The Society for Computer Simulation International, 1989.

Corey, E. J., *Pure Appl. Chem.* **14**, 19 (1967).

Dugundji, J., and Ugi, I., An algebraic model of constitutional chemistry as a basis for chemical computer programs. *Top. Curr. Chem.* **39**(19) (1973).

Evans, L. B., Boston, J. F., Britt, H. I., Gallier, P. W., Gupta, P. K., Joseph, B., Mahalec, V., Ng, E., Seider, W. D., and Yazi, H., ASPEN: An advanced system for process engineering. *Comput. Chem. Eng.* **3**, 319 (1979).

Henning, G. P., and Leone, H. P., "Design-Kit Users' Guide." Laboratory for Intelligent Systems in Process Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1990.

Meyer, B., Reusability: The case for object-oriented design. *IEEE Software*, March, pp. 50–64 (1987).

Nagel, C., Identification of hazards in chemical process systems. Ph.D. Thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA (1991).

Naur, P., Revised report on the algorithmic language ALGOL 60. *Commun. ACM* **6**, 1 (1963).

Nilsson, B., Object-oriented modeling of chemical processes. Doctoral Thesis, Department of Automatic Control, Lund Institute of Technology (1993).

Pantelides, C. C., SPEEDUP—recent advances in process simulation. *Comput. Chem. Eng.* **12**, 745–755 (1988).

Perkins, J. D., and Sargent, R. W., SPEEDUP: A computer program for steady-state and dynamic simulation and design of chemical processes. Selected topics on computer-aided process design and analysis. *AIChE Symp. Ser.* **214**, 78 (1982).

Piela, P. C., ASCEND: An object-oriented computer environment for modeling and analysis. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA (1989).

Piela, P. C., Epperly, T. G., Westerberg, K. M., and Westerberg, A. W., ASCEND: An object oriented computer environment for modeling and analysis: The modeling language. *Comput. Chem. Eng.* **15**(1), 53 (1991).

Sørlie, C. F., A computer environment for process modeling. Dr. Ing. Thesis, Department of Chemical Engineering, Norwegian Institute of Technology (1990).

Stephanopoulos, G., Artificial intelligence: What will its contributions be to process control? *In* "The Second Shell Process Control Workshop" (D. M. Prett, C. E. Garcîa, and B. L. Ramaker, eds.), p. 591. Butterworth, London, 1990.

Stephanopoulos, G., Johnston, J., Kriticos, T., Lakshmanan, R., Mavrovouniotis, M., and Siletti, C., DESIGN-KIT: An object-oriented environment for process engineering. *Comput. Chem. Eng.* **11**(6), 655 (1987).

Stephanopoulos, G., Henning, G., and Leone, H., MODEL.LA. A modeling language for process engineering. I. The formal framework. *Comput. Chem. Eng.* **14**(8), 813 (1990a).

Stephanopoulos, G., Henning, G., Leone, H., MODEL.LA. A modeling language for process engineering. II. Multifaceted modeling of processing systems. *Comput. Chem. Eng.* **14**(8), 847 (1990b).

Vernin, G., and Chanon, M., eds., "Computer Aids to Chemistry." Ellis Horwood, Chichester, 1986.

Wipke, W. T., Heller, S. R., Feldmann, R. J., and Hydes, E., eds., "Computer Representation and Manipulation of Chemical Information." Wiley, New York, 1974.
Woods, E. A., The Hybrid phenomena theory: A framework integrating structural descriptions with state space modeling and simulation. Dr. Ing. Thesis, Department of Engineering Cybernetics, Norwegian Institute of Technology (1993).
Woodward, R. B., *in* "Perspectives in Organic Chemistry" (A. Todd, ed.), p. 155. Wiley (Interscience), New York, 1956.
Woodward, R. B., *in* "Pointers and Pathways in Research" (G. Hofteizer, ed.), p. 23. Ciba of India, Ltd., Bombay, 1963.