Object-Oriented Programming Applied to Laboratory Automation. 2. The Object-Oriented Chemical Information Manager for the Analytical Director

T. Zhou and T. L. Isenhour*

Department of Chemistry, Kansas State University, Manhattan, Kansas 66506

J. C. Marshall

Department of Chemistry, Saint Olaf College, Northfield, Minnesota 55057

Received June 1, 1992

The Chemical Information Manager (CIM) has been designed and implemented as an object-oriented database management system for the Analytical Director. A conceptual laboratory model has been developed using object-oriented modeling techniques to abstract a robotic analytical laboratory, in which the CIM manages the data about chemical properties and chemical reactions. By interacting with the other modules, the CIM assists a user to design a analytical procedure and to validate and optimize the designed procedure. The CIM is capable of learning from interactive user input and the results of robot execution. The addition, removal, modification, and retrieval of data with a relational object model are also discussed.

INTRODUCTION

In this paper, we present the design and implementation of the Chemical Information Manager (CIM) for the Analytical Director. The goal of the Analytical Director is to develop an intelligent system that can design, test, modify, and execute analytical procedures. The CIM has been developed as an object-oriented database management system which integrates with the other modules of the Analytical Director to accomplish a deep-reasoning-based knowledge inference engine.

Figure 1 shows the architecture of the Analytical Director. A conceptual model of an analytical laboratory has been developed using a structured knowledge representation and object-oriented modeling techniques.[1,2] In the conceptual model, the real robotic table is abstracted by a computer representation of a *workbench*, and the laboratory activities are represented by a set of primitive operations. The workbench consists of a set of objects representing the laboratory devices, instruments, and robot(s). The laboratory activities are simulated by enabling the primitive operations to "perform" on the workbench to change its state. The state of the workbench is a combination of the states of all instrument and device objects. The purpose of the simulation is to assure a state of the workbench matching the real robotic table. Such simulation requires a system knowledge base containing information about chemical properties and chemical reactions. The Chemical Information Manager (CIM) and the Chemical Reaction Generator (CRG) are designed to manage the chemical information database and to simulate chemical reactions.

OBJECT-ORIENTED MODELING OF AN ANALYTICAL LABORATORY

Basic Concepts of Object-Oriented Paradigm. The design of the conceptual model and the overall performance of the developed system have been described in previous papers.[1,3] To describe better the design criteria and how the CIM is related to the other modules, it is necessary to present the object-oriented design of the Analytical Director expert system. Figure 2 shows the hierarchy of structured objects which models an analytical laboratory capable of performing gravimetric, spectrophotometric, and volumetric analyses. This section serves two purposes: discussing briefly the object-oriented knowledge representation and introducing the basic concepts of object-oriented programming and design.

The object-oriented paradigm is based on the following fundamental concepts:[4,5]

Each Real-World Entity Is Modeled by an Object. Each object is associated with a unique identifier. This *entity* can be a real object or an abstract concept. The state.space hierarchy in Figure 2 represents the laboratory instruments and devices. The analytical.operations hierarchy represents the 14 primitive operations defined for our present robotic laboratory.

Each Object Has a Set of Data Attributes and Methods. The set of data attributes and the set of methods represent the structure and behavior of an object, respectively. Figure 3 shows the definition of the *testtube* object. It consists of 10 data attributes and nine methods. The data attributes specify the state of a test tube. The methods specify the behavior of a test tube. For example, the reaction method of the testtube object embeds the procedural knowledge about chemical reaction(s) when new chemicals are added into a test tube.

Objects Sharing the Same Structure and Behavior Are Grouped into Classes. A class represents a template for a set of similar objects. Each object is an instance of some class. Take the testtube object as an example; it defines a class of objects. The number of test tubes on a robotic table would depend on the particular robotic laboratory. We specify that workbench has 30 test tubes. Instead of defining the data and method attributes for each individual test tube object, we define a *class* of objects, testtube (refer to Figure 3). Each test tube object is then created as an instance of the testtube class. As instances of the testtube object, the 30 test tubes testtube-1 to testtube-30 inherit all the data attributes and methods from the testtube class. An object class defines a template for a set of objects. Each instance duplicates the data and methods structure and can have its own values of the attributes.

A Class Can Be Defined as a Specialization of One or More Classes. A class defined as a specialization is called a *subclass* and inherits data attributes and methods from its *superclass*(es). Figure 4 shows the superclass–subclasses relationship of the analytical.methods hierarchy. The ana-

0095-2338/93/1633-0569$04.00/0

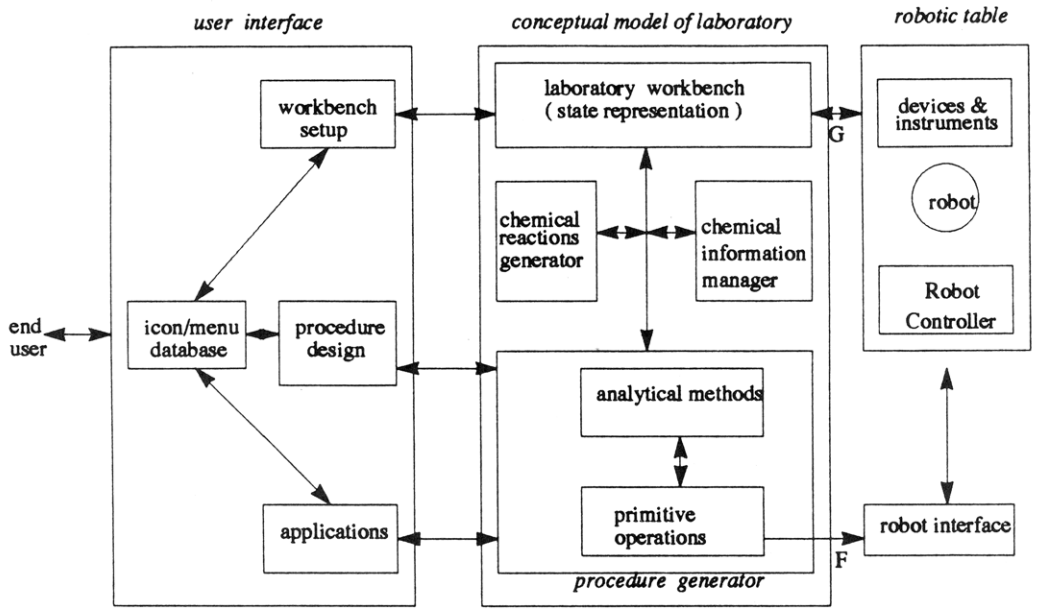


Figure 1. Architecture of the Analytical Director.

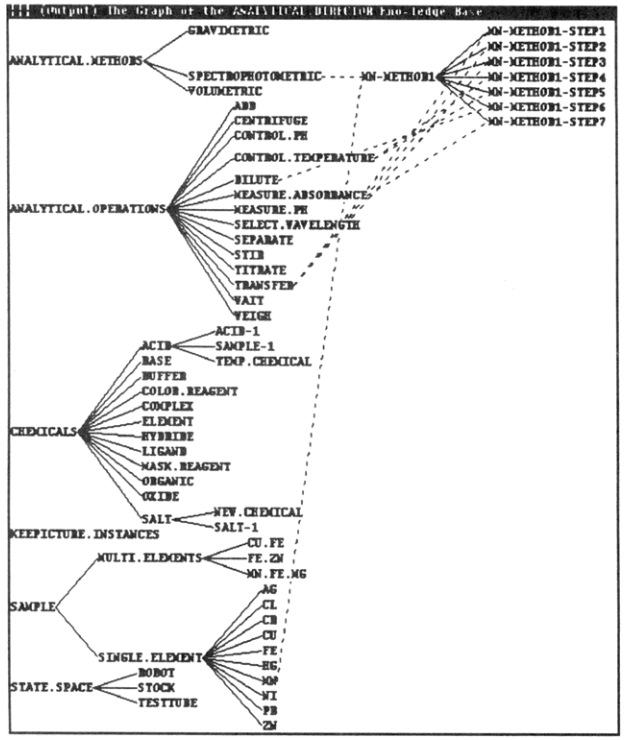


Figure 2. Object-oriented Modeling of a analytical laboratory.

lytical.methods object defines the attributes common to the three analytical methods like the data attributes where_is_-sample, where_is_standard, and target_element_state and the method attributes validate_procedure and generate_-robot_procedure. Three objects, gravimetric, spectrophotometric, and volumetric, are created as subclasses of the analytical.methods. As subclasses, each of these three objects inherits the data and method attributes of the superclass analytical.methods. Each subclass object also has its own data and method attributes. For example, the spectrophotometric object contains its own data attributes sample_-absorbance and standard-absorbance along with the inherited attributes from the superclass analytical.methods. Notice that the data and method attributes of the class analytical.methods are inherited by its subclasses. These inherited attributes of the subclasses are highlighted in Figure 4.

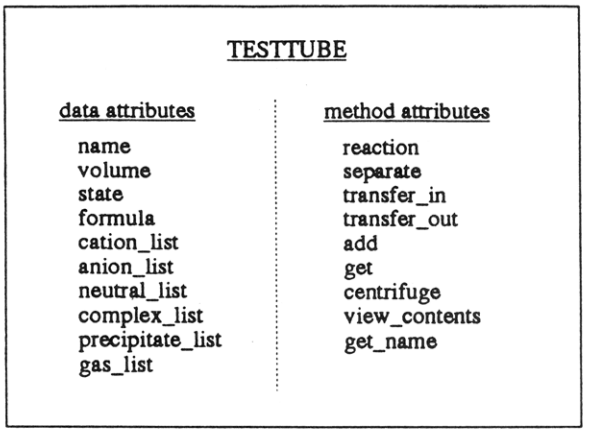


Figure 3. Definition of the testtube object.

Chemical Information Manager and Conceptual Laboratory. With an object-oriented model of the conceptual laboratory, designing a procedure is a process of creating object instances of certain defined object classes and activating the relevant methods. The object hierarchy of Figure 2 contains a procedure for quantitative manganese analysis.[3] The procedure is created as an object *MN-METHOD1*. The MN-METHOD1 is an instance of both *spectrophotometric* and *MN* because it is a spectrophotometric method for determining manganese. This method consists of seven steps. Each step is an instance of one of the primitive operation objects. With the inheritance, each step object can simulate the corresponding primitive operation and control the robot execution. The method object applies the operational principle of the corresponding analytical method to guide each step simulation, to validate a designed procedure, and to generate a complete robot procedure.

The laboratory simulation is achieved by enabling each step object to "operate" on the conceptual laboratory workbench, i.e., to change the state of the test tubes and stocks. The states of testtubes and stocks are mainly specified by the chemical components of their contents. In order to change the chemical components of a testtube upon adding some new chemicals, the system must have knowledge about these chemicals and the relevant chemical reactions. The CIM manages all the relevant chemical properties so that the expert

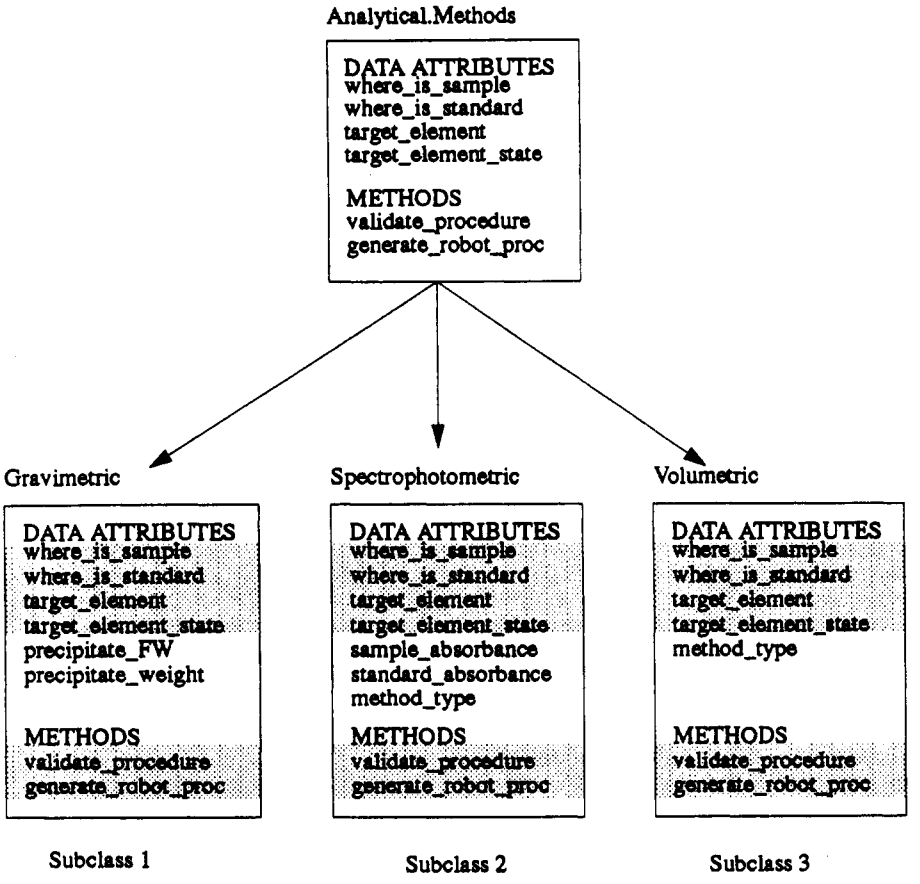


Figure 4. Superclass and subclass relationship.

system can recognize chemicals and simulate chemical reactions. More specifically, the developed CIM is able to do the following:

1. Retrieve the data efficiently. It recognizes the chemical type upon reading a chemical formula and then retrieves the relevant properties.
2. Retrieve the data intelligently. It knows, in the process of designing a valid procedure, what kind of information is needed. For example, if the user issues a command of "making solution of 0.1 M AgCl in 200 mL of water", instead of retrieving the name of AgCl or its chemical reaction properties, it looks for the pK_{sp} and finds out that it is impossible to make such solution because AgCl is insoluble in water.
3. Check the integrity before new data is stored into the database. For example, before a new acid is stored into the database, the CIM checks the properties of the acid to assure the consistency of the data record.
4. Manage and update the database in a dynamic way to reflect the experiments conducted by the robot. Each robot execution of a procedure could generate some new information. The CIM checks the recent information against the database, updates the data that should be changed, and inserts new data as they become available. Thus, the chemical information database becomes more complete.

A conventional relational database would not be adequate to support these capabilities due to its very limited semantic perception of the data and its lack of compatibility with an expert system.[6,7] The CIM has been developed as an object-oriented database management system. It interacts with the other modules to simulate an analytical laboratory, using knowledge of chemical properties and chemical reactions.

name	formula	state	pK_{sp}
silver chloride	AgCl	solid	7.80
silver bromide	AgBr	solid	10.3

DESIGN AND IMPLEMENTATION OF THE CIM

An object-oriented database management system (OO-DBMS) can be defined as a database management system that directly supports a model based on the object-oriented paradigm.[8] There are two basic approaches to OO-DBMS: extending a relational DBMS or extending an object-oriented programming language. The choice of approach would seem to depend on the application.[9] The CIM extends a relational DBMS with an object-oriented interface. Data are managed through standard object access. The CIM hides the details of a relational DBMS and removes some of the arbitrary restrictions.

Design of the CIM. A relational object model[10] is used in designing the CIM. In the relational object model, relations (data records) are represented as instances of objects. The integration of objects and relations is achieved by using a table to represent their structures. A relational database consists of a set of tables. Each table consists of columns (fields or attributes) and rows of data records. The example below demonstrates how such a table can be used to store chemical information.

A row in a table represents a relationship among a set of attributes. For the example above, the table consists of four fields (name, formula, state, pK_{sp}) and can be used to store a particular type of chemicals, *salt*. The definition of a table's fields and their domains is called the database schema. The pK_{sp} is the negative logarithm (base 10) of the solubility product constant of a salt. The attribute *name* is defined as the name of the salt with its domain to be a string value. The

Object:	SALT	
Parent:	relation	
Attribute:	Name	Value: *string*
Attribute:	Formula	Value: *string*
Attribute:	State	Value: *solid* or *liquid* or *gas*
Attribute:	pK_{sp}	Value: *numeral*

Figure 5. Relational object definition of a table.

Object:	ACID		
Parent:	relation		
Attribute:	Name	Value:	*string*
Attribute:	Formula	Value:	*string*
Attribute:	State	Value:	*set* (solid liquid gas)
Attribute:	Normality	Value:	*numeral*
Attribute:	Solubility	Value:	*numeral*
Attribute:	Mk_sln	Value:	*list of numeral*
		IF_added:	*check.mk_sln*
Attribute:	pKa	Value:	*list of numeral*
Attribute:	Integrity	Value:	*method (check_integrity)*

Figure 6. Object definition of the acid "relation".

attribute *formula* is defined as the formula of the salt, which takes a string to be its domain. The *state* is defined as the state of the salt with its domain to be either solid, liquid, or gas. This concept of a relational schema is analogous to an object template with attributes.[6] In the object-oriented model, "relations" are defined by a series of special objects, each having "relation" as one of its superclasses.

Figure 5 shows the object definition corresponding to the above table. An object class, *SALT*, is defined with four attributes and its superclass being "relation". Both the attributes name and formula take strings to be their values; the attribute state can be either solid, liquid, or gas; and the attribute pK_{sp} takes a numeral as its value. With the object class salt defined, each row of the table can be represented by an instance of the class.

In addition to the predefined types such as numeral, character, and string, an attribute may also be a set, a list, a method, another object, or a predicate. Figure 6 shows the definition of a more complex "relation" by an object. This object defines a class of chemicals, *acid*. The attributes name, formula, state, normality, and solubility define the corresponding properties of an acid. The mk_sln attribute takes a list of numerals as its value. It specifies how a concentrated acid solution can be used to make a diluted solution. The pK_a attribute takes a list of numerals. The first numeral of the list is the negative logarithm of the first dissociation constant of the acid, and so on. The attribute integrity takes a method as its value. The attribute mk_sln has an if_added predicate with an attached method check.mk_sln. A predicate associates certain procedural behavior to an attribute. For the above if_added predicate, the check.mk_sln method checks the correctness of the list of numerals. The if_added predicate needs to be approved before a new value can be assigned to the mk_sln attribute.

Adding and Removing Data. In the relational object model, adding data is done as follows: First, values are assigned to the attributes of an instance of the object. Second, the integrity of the new object is checked. Finally, the new object is added into the database. For example, sample-1 is an instance of the acid object (see Figure 2) with values as in Figure 7. After values are specified, the method check_integrity checks the consistency of the formula and the name; the increasing order of the members of the pK_a list, the correctness of the value of mk_sln list. Upon proving this predicate, then the object instance is stored into database.

Attributes of Child	Values
Name	"sulfuric acid"
State	liquid
Formula	"H2SO4"
pK_as	(null 1.84)
Normality	36.0
Mk_sln	((12 2.15) (6 5.08) (1 35.13))

Figure 7. Instance of the acid object class.

The data removal in the relational object model can be done with a predicate delete-instances, as shown in the following example.

```
Delete-Instances
from   acid
where  state = solid
```

This would delete all the instances of the acid with its state attribute being solid.

Retrieving Data. A predicate get-instances is used to retrieve information in the object model. The get-instances predicate incorporates a syntax similar to a structured qeury language[6] for retrieving information. Consider the following examples:

```
get-instance  'formula ID'
select        formula
from          salt
where         pKa > 5 AND
              (anion = "CrO4 2-" OR anion = "Cl-" )
```

The get-instance predicate selects, from the salt "table", all the insoluble ($pK_{sp} > 5$) chromate or chloride salts. It finds all the salts (a set) satisfying the retrieving condition, and then it binds the formula ID to the first one of the set. Subsequently, the formula ID binds to the next formula of the set until there are no more matches, and then the predicate fails. The *where* clause can be ANDed, ORed, and nested together to express any kind of searching condition.

Database Maintenance. In the relational object model, attached predicates can be used to ensure integrity maintenance and dynamic data management. One way to use an attached predicate is as a guardian for data retrieval and storage. It is mainly used to restrict the information being retrieved from or stored into an attribute.

As an example, consider the acid object defined in Figure 6. The attribute mk_sln has an associated predicate if_added. In Figure 7, values of a particular object (representing a commercially available concentrated sulfuric acid) are specified. The value of mk_sln is input to be [(12, 2.15) (6, 5.08) (1, 35.13)]. The semantic of the value implies "to make 12 N solution by mixing the concentrated acid with water in a ratio of 2.15; to make 6 N solution by mixing the concentrated acid with water in a ratio of 5.08; and to make 1 N H2SO4 by mixing the concentrated acid with water in a ratio of 35.13". Assume, using x to stand for the first number of each member of the list (i.e., the concentration) and using y to stand for the second value of each member (i.e., the ratio). Then, the if_added predicate would look like

```
if   (for_all_member_of_list: Normality is_close_to x*(y+1)
then approve
else reject
```

Such an if_added predicate would guarantee that the semantic is correctly maintained by the new value. To see if the concentrated commercially available acid (36 N for sulfuric

```
(defun select_db (thisunit expression last_logic)
    (let (operator)
     (setq operator (car expression))
    (cond(
        (equal operator 'AND)
        (intersection (select_db thisunit (cadr expression) 'AND)
                      (select_db thisunit (cddr expression) 'AND))
          )
        ( (equal operator 'OR)
          (union  (select_db thisunit  (cadr expression) 'OR)
                  (select_db thisunit  (cddr expression) 'OR))     )
        ( (and (> (list-length expression) 1)
                (listp  (car expression))
               (equal last_logic 'AND)    )
        (intersection (select_db  thisunit  (car expression)  'AND)
                      (select_db thisunit (cdr expression)  'AND))   )
        ( (and (>  (list-length  expression)  1)
                (listp  (car  expression) )
               (equal  last_logic  "AND)   )
        (union   (select_db  thisunit  (car expression)   'OR)
                 (select_db thisunit  (cdr expression)   'OR))  )
        (equal  (list-length  expression)  1)
                  ;;; This is the case for one simple logic expression
        (select_db  thisunit  (car expression)  'OR)     )
                                    ;;; it doesn't matter to use AND, OR
        ( t          ;;; If it is a simple logic expression  (not nested)
        (if  (equal  (second expression) '=)
            (findset  (string-downcase  (string  (pack*  "/usrb/txz/keedb/"
                      thisunit  (car expression)))))  (string  (third  expression))  )
          (getset  (string-downcase  (string  (pack*  "/usrb/txz/keedb/"
                      thisunit (car expression))))) (string (third expression))
                   (string  (second expression))          )
        )))     )
```

Figure 8. Function as the value of the select method of the chemical object.

acid) can be diluted to make 6 N H_2SO_4 by mixing the concentrated acid with water with a ratio of 1:5.08, calculate the value 6 × (1 + 5.08), which is approximately 36 and complies with the semantic.

Implementation. The CIM has been implemented as part of the system knowledge base (see Figure 2) of the Analytical Director. The system was developed on a SUN SPARC station under KEE (a Knowledge Engineering Environment developed by IntelliCorp). KEE is a knowledge system development product that provides software developers with a set of programming tools and techniques to build intelligent computer applications. The object-oriented interface to the database management is implemented with LISP language programming. The file and data record manipulations are implemented with C language programming. At the present time the database consists of 11 types of chemicals. They are acid, base, buffer, color.reagent, complex, element, hydride, ligand, mask.reagent, organic, oxide, and salt. Presently, the system designs procedures for volumetric, gravimetric, and spectrophotometric analyses. The chemicals encountered in these procedures are mostly inorganic. For this reason, a limited number of organic compounds (organic coordinating compounds are treated as ligands) are encountered, and they are stored in one common type, organic. When the system expands, this category will need to be further divided.

Corresponding to these chemical types, 11 objects are created. Each object has a set of data attributes specifying properties of the type and a set of methods to enhance the semantic requirement of the data attributes and to perform the dynamic data management. To reflect the specialty of these "relational objects", an object, chemical, is created as the superclass of these objects.

The chemical object has no data attributes but five method attributes. They are add-new, fetch, select, db-display, delete, and modify. All these methods are for the database operations. A method can be a block of LISP code or a function name. Figure 8 shows the LISP function select_db(thisunit expression last_logic) for the select method. This function has three arguments. The first one is bound to the selected object when it is activated. The second one is a logic expression for a

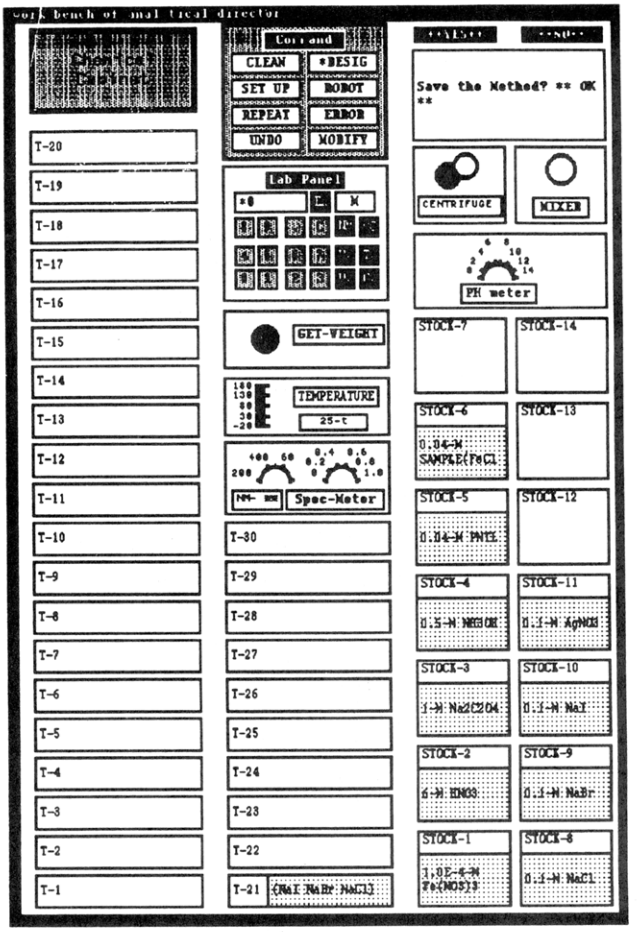


Figure 9. Pictorial representation of the workbench.

searching condition such as [AND (pK_{sp} > 2.0) (cation = "Ag")]. The third one is for the purpose of recursively calling itself so that nested logic expressions can be processed. As the subclasses of the chemical object, these 11 types of chemicals inherit all the five database operation methods from teh object chemical. So do the instances of these 11 objects. Suppose a new acid is encountered in a procedure design process. An object acid-1 is created to represent the new chemical; its chemical properties are specified. When the user needs to store the record of acid-1 into database, the add-new method of the acid-1 object is activated, which checks the properties of the acid for consistency and saves the new chemical into the database.

USE OF THE CIM

Chemical Cabinet. As an application of the CIM, we have constructed a chemical cabinet module for the conceptual laboratory model. A user can use the *chemical cabinet* to manage the data about chemical properties and chemical reactions. A user can also simulate design activities to transfer chemical reagents from the chemical cabinet to the workbench or from the workbench to the cabinet. Figure 9 shows the icon-based graphical representation of a laboratory workbench.[5] The chemical cabinet icon is the user interface to the chemical information management activities. A user clicks on the chemical cabinet icon to bring up its main menu (see Figure 10a). This menu contains four menu choices: chemical reaction, add new chemicals, update chemicals, and EXIT. The chemical reaction choice is used to store or modify chemical reactions. The add new chemicals can be used to add a new chemical to the database, which is useful when the

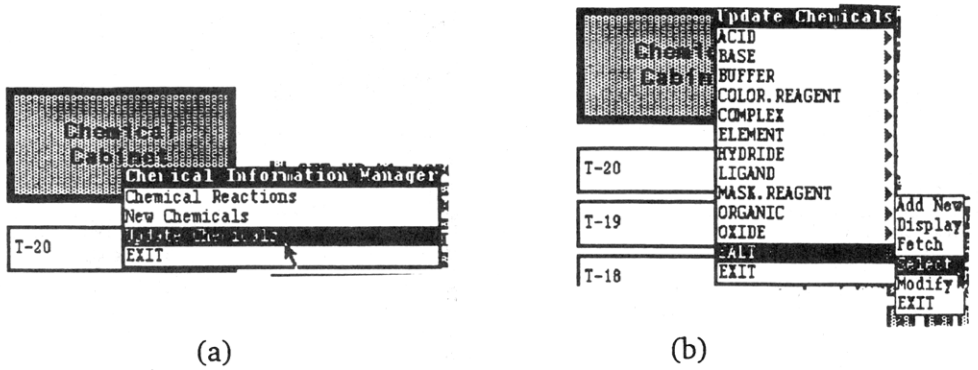


Figure 10. Icon manipulations to open the chemical cabinet for chemical information management.

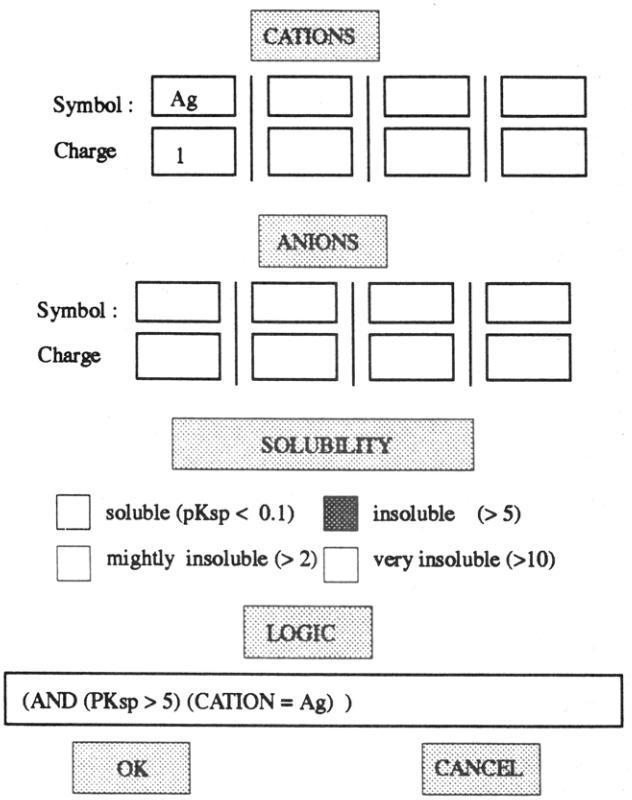


Figure 11. Specify the searching condition to select chemicals.

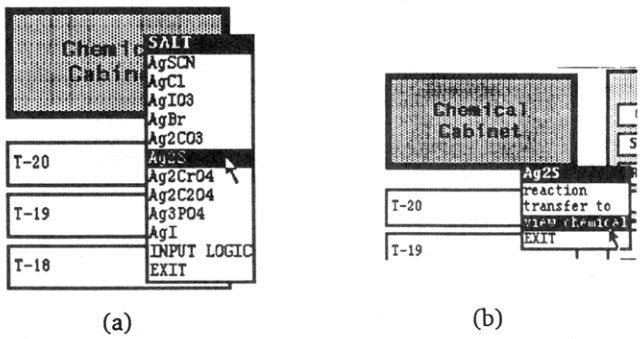


Figure 12. Chemicals selected and operations applicable to a chemical.

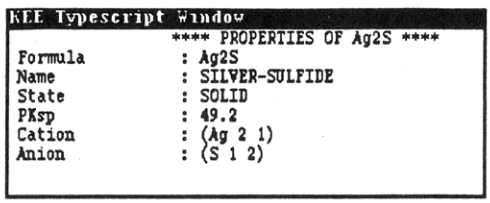


Figure 13. Properties of the chemical Ag_2S.

```
(7.05  ((1 "Ag" C 1) (1 "I"  A 1))   ((1 "AgI"  P 0)) )
(5.15  ((1 "Ag" C 1) (1 "Br" A 1))   ((1 "AgBr" P 0)) )
(4.85  ((1 "Ag" C 1) (1 "Cl" A 1))   ((1 "AgCl" P 0)) )
```

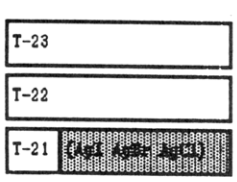
Figure 14. System representation of the three chemical reactions.

user does not have much knowledge of the new chemical. The update chemicals is used for data retrieval and modifications.

Suppose the user selects the update chemicals choice. This would bring up a menu containing all the chemical categories the current database contains (see Figure 10b). The categories are in an alphabetical descendant order with "update chemicals" as the title. Each selection generates a submenu of add_new, display, fetch, delete, select, and modify. The add new is for adding a new chemical into the selected chemical type. The display is for displaying all the chemicals of the corresponding chemical type. The fetch is for retrieving properties of a particular chemical. The delete is for removing a chemical from the database. The select is for finding chemicals with specified searching condition. The modify is for updating properties of a chemical. Suppose a user selects the menu choice of salt and further chooses the select submenu. The system then displays a window (see Figure 11) for the user to input the search condition. Because the chemical type is salt, the user can use the cation and (or) the anion and (or) the solubility to specify the searching query. Suppose a user wants to search "all the silver salts which are insoluble".

Figure 11 shows two methods to specify searching conditions. One method is based on chemistry notations in which a user specifies the cation, anion, and solubility. For the above example, a user can simply specify the cation to be Ag^+ and specify the solubility to be insoluble. Another method is to use *AND–OR* logic expressions. The logic expression for the above example is shown at the bottom in the figure. The chemical notation method is straight-forward to a chemist, but the logic expression method is more flexible, especially for a complex searching condition. When a chemist uses the chemistry notation method to specify the search query, the corresponding logic expression will be automatically shown at the bottom. After specifying the search condition, the method select of the selected chemical type salt is activated. The inherited method select is activated with the argument "[AND (CATION = "Ag[1]") ($pK_{sp} > 5.0$)]". The returned list of chemicals is used to form a new menu. The new menu contains all the chemicals selected (refer to Figure 12a).

If the user selects a chemical, for example, *Ag2S*, from the menu of Figure 10a, then a menu with all the possible operations applicable to the selected chemical pops up as in Figure 12b. The reaction is for modifying the existing chemical reaction information or adding new chemical reaction information relevant to the chemical. The transfer to choice can



(a)

```
KEE Typescript Window
                  ***** CONTENTS OF TESTTUBE-21 *****
Formula           : (AgI AgBr AgCl)
State             : MIXTURE.MIXED
Total volume      : 7 ml
Cation_list       : ((0.3 Na C 1) (0.1 Ag C 1))
Anion_list        : ((0.4 No3 A 1))
Precipitate_list : ((0.1 AgI P 0) (0.1 AgBr P 0) (0.1 AgCl P 0))
```

(b)

Figure 15. Graphical representation and literal description of the result of a chemical reaction simulation.

```
KEE Typescript Window
              ******    FE-METHOD1    ******
step  1 : transfer 2 mL of SAMPLE(FeCl3) from STOCK-6 to
          TESTTUBE-1
step  2 : transfer 1 mL of 6-M HNO3 from STOCK-2 to
          TESTTUBE-1
step  3 : transfer 1 mL of 0.5-M NH3OH from STOCK-4 to
          TESTTUBE-1
step  4 : transfer 2 mL of 1-M Na2C2O4 from STOCK-3 to
          TESTTUBE-1
step  5 : transfer 1 mL of 0.04-M PNTL from STOCK-5 to
          TESTTUBE-1
step  6 : dilute SOLUTION(FePNTL X 2) at TESTTUBE-1  to 10 mL
step  7 : stir the sample SOLUTION(FePNTL X 2) at TESTTUBE-1 for 2 min
step  8 : measure absorbance of SOLUTION(FePNTL X 2) of TESTTUBE-1  at 510 nm
```

Figure 16. Designed procedure for spectrophotometric determination of iron.

be used to transfer a certain amount of the chemical to a stock or a test tube. Selecting the view chemical choice displays the properties of the selected chemical (Figure 13).

Chemical Reaction Simulation. During the process of designing a procedure, the expert system simulates the chemical reactions associated with each designed step and updates the state of the workbench. The state of the workbench determines what kind of operations are applicable. For example, it is not legal to transfer 1 mL of solution out of testtube-1 while the test tube contains a mixed precipitate. The chemical reaction simulation depends on two knowledge sources: data about the chemical reactions and a set of heuristic rules to generate possible chemical reactions.

We use an example to show how thermodynamic data such as pK_{sp} can be used to predict chemical reactions. With a state of the workbench as in Figure 9, we design a primitive operation of "transferring 4 mL of 0.1 M $AgNO_3$ from stock-11 to testtube-21". The testtube-21 contains 1 mL 0.1 M of each NaCl (transferred from stock-1), NaBr (from stock-2), and NaI (from stock-3).

The real situation is that several chemical reactions would occur simultaneously. But as a computer simulation, the possible chemical reactions are "performed" in a sequential order. The CRG checks all the possible chemical reactions between eight pairs: Ag^+&Na^+, Ag^+&Cl^-, Ag^+&Br^-, Ag^+&I^-, NO_3^-&Na^+, NO_3^-&Cl^-, NO_3^-&br$^-$, and NO_3^-&I^-. The CIM supplies the relative thermodynamic data to the CRG, and the CRG determines all the possible chemical reactions. In this case, only three precipitation reactions are found possible based on the pK_{sp} values of AgCl, AgBr, and AgI. Figure 14 shows the system representation of the chemical reactions generated by the CRG.

Each chemical reaction is represented by a list of three members, two of which are lists. The first is the reaction possibility which determines the sequential order of the reaction being simulated. The second is the reactant list. The third is the product list. For precipitate reactions, the pK_{sp} is used to calculate the reaction possibility. Consider the first reaction, the first member 7.05 specifies its reaction possibility; the second member [(1 "Ag" C 1) (1 "I" A 1)] and the third member [(1 "AgI" P 0)] imply that 1 mol of cation Ag^+ (system representation: "Ag" C 1) reacts with 1 mol of anion I^- ("I" A 1) to produce 1 mol of precipitate AgI ("AgI" P 0). Figure 15a graphically shows the state of testtube-21 after the chemical reaction simulation. Figure 15b shows a description of the components of the testtube-21 as a result of the chemical reactions.

Self-Improvement of Chemical Information Database. The CIM is capable of self-learning new chemicals. There are two sources for the Analytical Director to acquire knowledge: the chemists who design procedures and the robot which executes designed procedures. An explanatory and suggestive mechanism is used to learn new chemicals during a design process. Suppose a user needs to convert the following English description to a robot executable procedure.

> Pipet 1.0 ml of unknown sample into a 10-ml test tube. Add 1.0 ml of saturated sodium acetate solution, 1.0 ml of the 10% hydroxylamine hydrochloride solution, wait for 5 minutes, and add 1.0 ml of the 0.1% 1,10-phenanthroline solution. Allow 10 minutes, dilute to volume, and read the absorbance at 510 nm using water as a blank.

At the configuration of the laboratory workbench of Figure 9, a chemist might design the procedure as in Figure 16.

As the chemist assigns each chemical reagent and designs each step, the system interacts with the user and activates the CIM from time to time to recognize a chemical being input by the user. Suppose the CIM does not contain the data record of the chemical 1,10-phenanthroline. When the chemist assigns 1,10-phenanthroline solution to stock-4, the system recognizes it as a new chemical.

The system then starts the learning process for the new chemical. Figure 17 shows how the system prompts the user to input the chemical properties and chemical reactions about the new chemical. The *abbreviation* is a system representation for some organic compound which has a relatively complex formula. Such an abbreviation simplifies the notation of a chemical reaction equation. The abbreviation of 1,10-phenanthroline is PNTL. The chemical type ligand specifies into which table (or database file) the new chemical will be

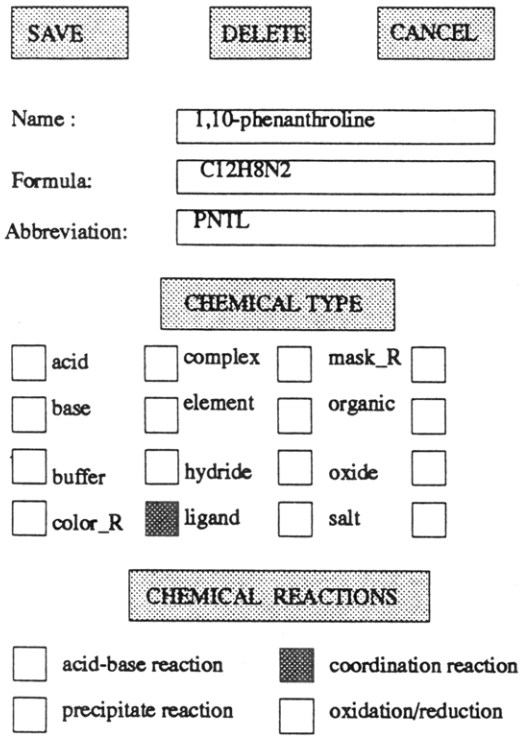


Figure 17. Learning of a new chemical.

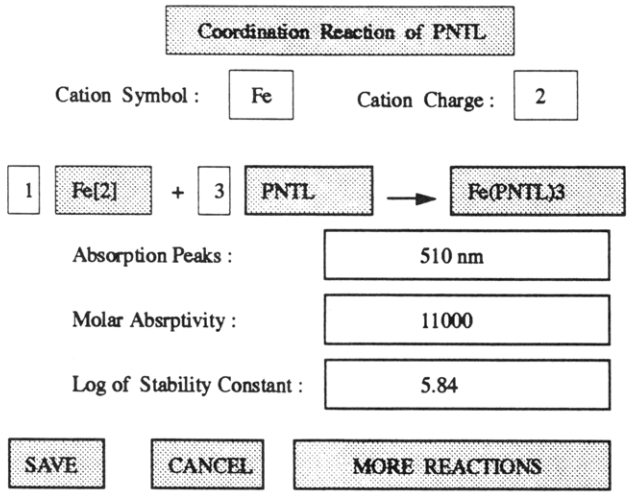


Figure 18. Learning of a coordination reaction.

saved. There could be many chemical reactions associated with a new chemical. It depends on the chemist to teach the system the important chemical reactions. For 1,10-phenanthroline, the chemical reaction involved in this procedure is the coordination reaction with iron(II). The user selects the coordination box for coordination reactions. Figure 18 shows the window display for a user to input the coordination reaction between 1,10-phenanthroline and ferric icon (Fe^{2+}). The system abbreviation PNTL is used to express the chemical reaction equation. After the user specifies the cation and the number of the ligand, the product of the reaction is automatically shown. The user can then specify the absorption maximum peak and the corresponding molar absorptivity of the product. The user is also prompted to input the stability constant of the product.

After the above learning process, the new chemical properties and the chemical reactions become part of the knowledge base. For the above example, the system interacts with the user and starts the learning process when the user assigns each stock solution. When it is time to design each of the steps in Figure 15, the system uses the newly acquired knowledge together with the old database to assist the design process, to validate the procedure, and to simulate the robot execution.

Each robot execution could generate certain new information which is used by the expert system to update the database. For the procedure above, the robot execution generates the following data: the color development time of the method, the linear concentration range, the molar absorptivity of the color complex, the samples analyzed, and the execution time. These data are stored into corresponding object classes. For example, the molar absorptivity 11 000 of the complex Fe-$(PNTL)_3$ would be stored into the corresponding $Fe(PNTL)_3$ object instance of the complex object class.

CONCLUSIONS

An object-oriented database management system CIM has been designed and implemented for the Analytical Director project. With the relational object model, chemical information can be added, deleted, retrieved, and updated. As an object-oriented approach, the CIM enhances the semantic aspect of a data attribute and extends the dynamic database management of a conventional relational database. An SQL-like syntax is used by the CIM to selectively search for chemicals which is important for the Analytical Director in the process of designing new procedures. The CIM is essential for the automatic update of the database. The CIM together with the CRG has allowed the conceptual laboratory model to simulate chemical reactions. With the chemical reaction simulation, the expert system can keep track of the state of the workbench and compare it with that of the real robotic table to validate a designed procedure and control robot execution.

Chemical properties formula, state, pK_a, pK_b, pK_{sp}, standard reduction potential, stability constant, spectrophotometric absorption peaks, and absorption sensitivites for some common chemicals have been input into the database. These data have been used successfully by the expert system to convert some textbook analytical procedures and to design new procedures. Information about specific chemicals and chemical reactions has been acquired by a learning mechanism during the design process.

REFERENCES AND NOTES

(1) Bleyberg, M. Z.; Zhou, T.; Isenhour, T. L.; Marshall, J. *The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*; July 1990; The Design and Implementation of an Analytical Chemistry Expert System. ACM Press: Charleston, SC, 1990; pp 1073–1078.
(2) On Concurrency Control in an Analytical Chemistry Expert System. Zamfir-Bleyberg, M.; Zhou, T.; Isenhour, T. L.; Marshall, J. C. *Int. J. Appl. Artif. Intell.*, accepted for publication.
(3) Object-Oriented Programming Applied to Laboratory Automation. 1. An Icon-Based User Interface for the Analytical Director. Zhou, T.; Isenhour, T. L.; Zamfir-Bleyberg, M.; Marsshall, J. C. *Chem. Inf. Comput. Sci.* **1992**, *32*, 79–87.
(4) Boock, G. *Object-Oriented Design*; Benjamin/Cummings Publishing Co.: Reading, MA, 1991.
(5) Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. Object-Oriented Modeling and Design; Prentice Hall: New York, 1991.
(6) Date, C. J. *An Introduction to Database Systems*; Addison-Wesley: Reading, MA, 1990.
(7) A Case Study of the Difference Between Relational and Object-Oriented Database Systems. *Proc. OOPSLA '87* **1987**.
(8) Bertino, E.; Martino, L. Object-Oriented Database Management Systems: Concepts and Issues. *IEEE Comput.* **1991**, *24* (4), 33–48.
(9) Premerlani, W. J.; Blaha, M. R.; Rumbaugh, J. E.; Varwig, T. A. An Object-Oriented Relational Database. *Commun. ACM* **1990**, *33* (11), 99–109.
(10) Parsaye, K.; Chignell, M.; Khoshafian, S.; Wong, H. *Intelligent Databases*; John Wiley: New York, 1989.