A New Modular Architecture for Chemical Structure Elucidation Systems

Igor I. Strokov*,† and Konstantin S. Lebedev

Novosibirsk Institute of Organic Chemistry, Siberian Division of Russian Academy of Sciences, Lavrentiev avenue 9, Novosibirsk 90, Russia

Received December 7, 1995⊗

A highly modular Windows-based approach for development of complex systems is proposed under the name of *mosaic* or *artel* architecture. A mosaic system is defined as the unity of a set of independent applications representing chemical objects and separately their features and a set of oriented *links* describing a mode of interactions between applications. Following this definition a *mosaic document* concept is introduced in contrast with a "compound document" in OLE. The capability to split a programming burden into distinct tasks and to carry out the majority of the system's functional filling on the level of interapplication links are the main features of the artel architecture.

BACKGROUND

As computer systems evolve they tend to become larger and more complex—at least reverse examples are very rare. The problem is that the labor-consuming character of a final product usually grows much faster than its size and complexity. Many contemporary programs are criticized for excessive volume not adequate for their functional filling. Large programs are difficult for development and support and are awkward in use as they take up a lot of resources, require much time for loading, and are often replete with errors.

How can the convenience and reliability of large systems be enhanced? A key to this problem is known to lie in a modularity concept, i.e., in system division into separate substantive components (modules). There are several implements including program languages on one hand and operational systems (OS) on the other aimed to achieve modularity. It should be noted that despite significant progress in computer science and existence of such advanced languages as Ada, C and C++ bearing no built-in facilities of program linking and communication of processes dominate there. Hence in these points one should rely only upon an operational system. This course of things can be seriously criticized but can hardly be changed in the near future. Thus, considering modularity on the level of distinct programs one can lose sight of a program language and stay at the ability level of OS to support existence and interaction of many processes.

It is quite evident now that the benefit of multitasking is not brought down to the ability of many persons to use a single computer. Multitasking offers big opportunities in designing complex program systems. The simplest method means running, of parallel processes in the background, usually computational programs capable of exchanging data via files and claiming no graphic interface. Traditionally this method is propagated in a UNIX environment. A well-known commercial system of molecular design conjoining high-level graphics and heavy computations can be adduced as an example.[1]

The other modular approach known as dataflow has been carefully explored in the field of scientific visualization.[2−4] In brief, the dataflow concept may be described as a visual extension of the UNIX pipes and filters mechanism. To become an image raw data have to pass through a set of convesion modules whose input and output ports may be connected in a proper way. The LinkWinds[4] system has such modules and, moreover, control facilities like menus and sliders arranged as distinct applications, that are common to our approach.

Consequent usage of UNIX features for program integration in the field of chemistry has been proposed and implemented by Ihlenfeldt and co-workers in WODCA[5] and later in CACTVS.[6] Networking features and broad-scaled exchange of data and algorithms are the substantial parts of the last project. The ideas of the present paper occupy a narrower scale as they bear upon a local system. The spectral system for chemical structure elucidation developed by the authors and named ChemArt[7] is discussed here.

However the present paper deals with only OS Windows and Windows-based decisions. Not only is the choice of this OS dictated by its wide expansion—being quite a modern system. Windows provides some specific features which are important for the proposed architecture.

The most famous method of programs interaction in Windows is Object Linking and Embedding (OLE).[8] It allows one application to use objects created and processed in another. Not dwelling in details note that here, similar to UNIX examples, the matter is rather integration of ready and self-sufficient programs more than a way of designing them. The use of OLE allows one to apply programs widely and with more comfort, but one can hardly find a case in which OLE would help a system to be more compact.

An outward property of the decision proposed here is that the system is constituted from small, but numerous applications (now there are about two dozen of them ranging from 3 to 60 KB and answering to chemical objects and applied methods). Although all of them are absolutely viable singly, real problems require a group of several tens of application instances to interact with each other. Further, ChemArt can be qualified as a polymorphic system, since it has no single recognizable front. A front and a functional filling of the system are defined by a group composition and its members interaction. These factors, in turn, depend on a problem type.

† E-mail: strokov@dpsl.nsu.nsk.su.

⊗ Abstract published in *Advance ACS Abstracts,* June 15, 1996.

S0095-2338(95)00180-6 CCC: $12.00

Table 1. The Link Table for the Scene Presented in Figure 1

link no.	first appl	second appl	link type
1	executor	director	*boss*
2	director	executor	*subordinate*
3	executor	mass spectrum	*argument*

The features mentioned are surely insufficient for comprehension of the proposed architecture. Applied systems often consist of many interacting applications. It all depends on the way of roles division between applications and on the way of their interaction. To ascertain this question let us consider the following example.

INTRODUCTORY EXAMPLE

Let us take a computer system in the area of mass spectroscopy. Let a procedure of reducing a mass spectrum to the monoisotopic form require inclusion in the system in a way allowing a user to call the procedure up from a menu and watch the result (Figure 1).

If the system is a monolithic program already capable of drawing a mass spectrum, then the obvious decision comprises four steps:

1. add a new item to the menu;
2. add handling of the new menu command (call of a subroutine) in the body of the program;
3. add a module containing the subroutine to the system's project;
4. rebuild the executable module of the system.

In the ChemArt system nothing goes on in this way. Above all, there is no monolithic application. The menu owner (let us call it a *director*), the viewer of a mass spectrum, and the executor of a command (the function owner) are three different applications. If the first two applications already exist, then integration of a new facility is reduced to creation of a new application capable of nothing but making a reduction to the monoisotopic form. In practice it is the function itself arranged as a distinct task. All three applications are to be run simultaneously and to be informed as to each other's existence and roles. This information is presented as a *link table*, where a *link* is a ternary of numbers ciphering what application is bound with which one and what is a matter (or a type) of link.

In our case three links numbered 1−3 on Table 1 are sufficient for the given task. From link 1 the executor recognizes the director and inserts *Monoisotope* item in one of director's pop-up submenus (say *Tools*). On receiving a command corresponding to *Monoisotope* the director hands it to the executor, which is known to him due to link 2. The executor, having information on the mass spectrum due to link 3, processes it and makes it redraw. We shall return to the circumstances of these proceedings later and now introduce some terms.

Let the community of applications capable of joint work be called *artel* (with the soft 'l'), that stands for a traditional cooperation form in Russia. Let a unity of three items, a set of artel members, their appearances, and a link table, be called *scene* (Figure 1 represents an example of a scene). An artel-based architecture gets an adjective *artel* or *mosaic*.

The introductory example already illustrates the main features of the artel approach. Let us discuss these features more closely.

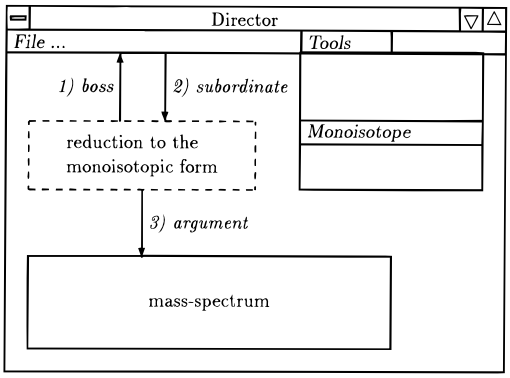


Figure 1. The scene of reducing to the monoisotopic form.

DIVISION OF CONTROL ACTIVITIES

The task of our mental experiment was confined in addition to a new function in the system. Handling of a function primarily implies storage of its code and data. On the other hand, an owner of a function is to properly arrange its call in corresponding circumstances. This activity has a control character. Implementation of the storage activity increments only the size of a program, whereas the control one influences both the program size and logical complexity.

Obviously a monolithic application handles all its functions however numerous they are. By virtue of dynamic linking and disk swapping, modern operational systems allow a programmer almost not to take care of the application size. That is, it does not require special efforts to make an application distributed in the sense of its code and data bulks. But as all the logic of a task is still in one hand, a "monolithic" program remains such in the sense of control activities division.

Another seemingly more fair principle acts in an artel: all the activities around a function are held by its owner. Now as long as it manages a function it should also call the function and attend to its arguments. Other applications, including argument owners cannot suspect its existence. It does not connotate that applications are not concerned with a collaboration. This opportunity is provided by the following two points:

1. The information on interrelations of applications (a link table) is freely accessible.
2. All the artel members undertake to duly respond to a definite set of messages being sent from one application to another.

The convention on the link table representation, on the implication of messages, and on the manner of responding to them is called an *artel protocol* (AP).

ARTEL PROTOCOL

Let us return to our example to illustrate both parts of the AP.

Let the owner of a mass spectrum keep it as an integer array *ms*[ ], where *ms*[*i*] is the intensity of a peak with *m*/*z* = *i*, and let the reduction function use an address of this array as a single argument. Before making a call to the function its owner must know this address. These data acquisitions require two steps:

1. recognize an owner of a mass spectrum;
2. inquire of the owner for an array address.

The implementation of the first step is quite evident: the function owner (executor) looks through the link table seeking a link with *argument* type and with the first application equal to itself (in our example it is link 3 in Table 1). To accomplish the second step the executor sends some special message to the mass spectrum owner. The last has to return the address of *ms*[ ] as an answer to this message. Further the executor can use this array as its own although it belongs to another application. The lack of interprocess protection in Windows is favorable here—another OS would make data exchange more tricky.

As a rule, details such as looking in a link table or answering messages are hidden from a user by means of a set of functions encapsulating the AP. It should be noted, however, that other messages besides special (or *user*, in Windows terminology) messages require a protocol answer. Consider the following example: let function *monoisotope* just mark isotopic peaks by a color instead of removing them. The equity requires this work to be entrusted to the function owner itself instead of a mass spectrum one. Otherwise diversity of functions processing a mass spectrum would lead to over-numerous and perhaps dissonant demands to its owner's drawing abilities. So an owner of an argument of any kind allows all applications linked with it by specific *paint* links to patch up an argument's figure by sending another special message to these applications in the end of its own drawing routine.

The total number of messages participating in relations between function owners and other applications equals four, whereas variety of functions in the system can be much larger. An analogy with a child meccano comprising a lot of details but very few types of connections to assemble these details is appropriate here.

MASTERS AND SERVANTS

Meccanoes (e.g., "Lego") resemble an artel in one more point: usually they contain many simple details of few kinds and lesser amounts of more elaborated and diverse ones (such as people's figures). Similarly the artel includes very few owners of "corporeal" objects, such as a structural formula (chemical structure), a mass spectrum, or other kinds of molecular spectra, but all these objects are used generally as input or output arguments almost in any problem from the given field of sciences. Let us call the owners of all the mentioned objects *servants* since they serve objects by drawing and editing them, in such a way that other applications, so called *masters*, are delivered from this business.

Not dwelling in the sense of this division one can just enumerate all the servants and relegate other artel members to masters (Figure 2). In this case the last category obtains very dissimilar applications in which the most extended group is composed of so called *procedural masters* embracing the executor from the introductory example. As it follows from the appellation the main goal of these masters is to implement functions (procedures) and to maintain interfaces to them. In distinction from servants procedural masters are almost always invisible and have only a pop-up dialog window allowing one to view and correct function options. The term "options" is applied here to simple

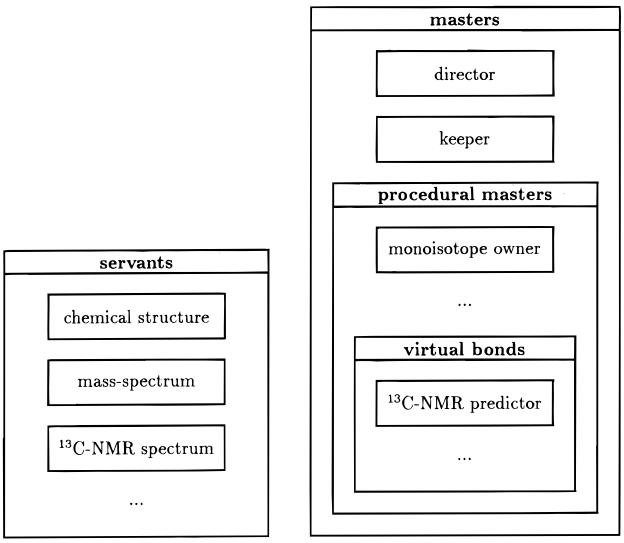


Figure 2. The classification of artel members.

parameters which affect function working and are situated in its owner itself. The denomination "arguments" in this case remains valid for input and output function parameters belonging to other applications. A number of arguments in abstract is not limited and ranges in reality from 1 to 6.

Calling methods may also differ. A call of a function on a menu command (see the introductory example) is convenient for simple functions without options. Otherwise the call of a function may be substituted by emersion of a dialog window containing controls for options and a starting push button. However, in both cases a user does initiate the action. Let us consider another method requiring no human intervention.

VIRTUAL LINKS

In the artel an automatic call is easily realized by means of *notify* links going (despite *argument* links) from an argument owner (usually a servant) to a function owner (master). By virtue of these links a servant is aware of masters it serves as an argument and is able to notify them about any modifications befallen to him by means of one more kind of message. A master receiving this message is free to treat it in any way. Usually a master is expected to execute its function and affect output arguments. But evidently such reaction is needless in some generic cases, e.g., if the message emanates from an end-product of a current function. Therefore *notify* links are never set automatically as the inverse of any *argument* link. Every master sets *notify* links selectively only for the argument owners it would like to get messages from. The same is also true for the previously mentioned *paint* links.

The next example illustrates this mechanism. Let the system have a list of structural hypotheses of an unknown compound, its experimental ^{13}C NMR spectrum, and a function predicting this kind of spectrum for a structure. The purpose at hand is to make a choice of a hypothesis to cause appearance of a hypothetical spectrum for visual comparison to the experimental one. The scene actualizing this goal is shown in Figure 3.

The ^{13}C NMR spectrum prediction using correlation tables is fast enough to perceive no delay between structure and spectrum appearances. A spectra comparison function called

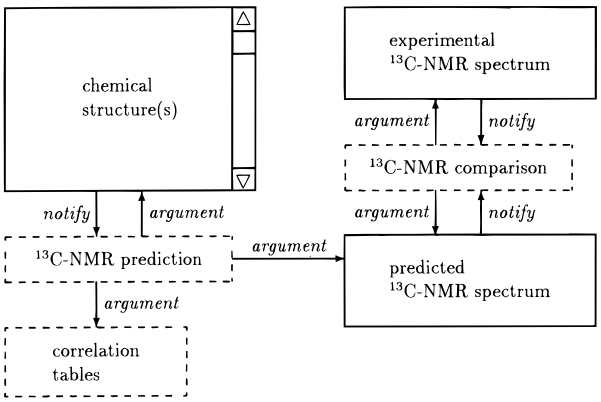
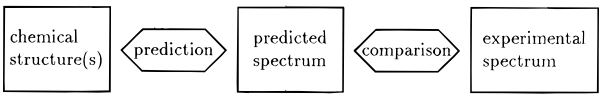


Figure 3. The scene for ^{13}C NMR spectra prediction and comparison.

automatically by a change of any of its arguments fulfills the scene. If both owners of the functions are hidden (that is fully authorized in this case), then the beheld result can be interpreted as the existence of special links between three visible windows:



Let such "logical" links realized by virtue of procedural masters (in distinction to *elementary* links) be called *virtual*. Virtual links as compared to elementary ones reflect in a more abstract level of the artel members interaction. The fact that a number of nodes in a virtual link is not limited by 2 and a link type has a plain chemical meaning expedites formation of complex scenes containing tens of applications and constituting real ChemArt system.

DIRECTION

The direction of scenes is held by a special master application called, in parallel to the theater, a *director*. It should be kept in mind that a scene description includes three chapters:

1. set of applications;
2. link table;
3. appearances of participants.

Usually all these data are collected in a document called a *scene script*. Its first two chapters replay for functional contents of a scene, whereas the last chapter allots a given functional set of different fronts. For each member of a scene the third chapter describes some application features allowable within a given OS: windows disposal, their visibility, presence of menus, captions, frames, etc.

A director is called for its capacity both to read a script and compose the last from an existing scene. Director's actions aimed at the loading of artel applications and sustaining the system integrity impression are quite routine and free from any artel tricks. A script format and its drafting of a given scene do not carry abstract difficulties too. Digressing from these details one can consider a director to be an editor of specific *mosaic documents* composed from separate applications. Let a mosaic document be a set of scripts realized as a consecution of scenes. This view should be noted to refer closely to the OLE *compound document* concept. Indeed a director can be compared to an OLE client and artel servants—to OLE servers. However, besides the technique of implementation there are grave differences which can be reduced to several items:

1. There is no distinction between an object and its server in our approach. A servant application is an object itself.
2. Artel objects and their properties are apportioned by separate applications.
3. The interaction of objects is a part of a mosaic document and do not agitate its owner (a director).

To the last point one can add that though a director is free from the duty to monitor objects' interactions, there is no taboo on doing it. Director's interference in scenes acting and alternation is sometimes the best way to tune the whole system operation and appearance.

KEEPER

One more factor of a mosaic document can be appreciated after the final point in the classification is considered—a *keeper* application belonging to the class of masters (see Figure 2). As it follows from the appellation a keeper implements input and output operations from long-term stores, i.e., disk files. All the other artel members, especially servants, try not to do it by themselves trusting on the interaction with a keeper via AP links. Let us designate the main reasons for it.

First, data exchange in ChemArt is based on a relatively complex index-sequential access method (ISAM) which affords to add, delete, and update records and to find them by their contents (keys). Not reaching the level of relation databases ISAM is still too elaborate to be used by any application without sacrificing its dimensions essentially. Not only the volume of ISAM functions but also auxiliary user and program interfaces should be accounted. That is, in designing an application one should not bother at all to include such standard facilities as *Open* and *Save*, since they can be later put to a ready program by means of keeper links.

Second, data exchange on the AP level gives independence in a way of ISAM functions implementation by a keeper. Thus any change in this way would involve only single application—a keeper.

Now a keeper uses an original file handler named "JoKey" which features can be briefly outlined in the following paragraphs.

Disk memory saving. JoKey provides for records and keys of variable length and the "context shrinkage" of keys. The last method means casting coincident ends or beginnings in every pair of successive keys away. Context shrinkage is very efficient in tables with compound or string keys.

The "limitless" cache. JoKey is able to use *all* the free computer memory for input/output cache without any sacrifice of other applications. Thank to two-level memory requests in Windows and existence of discardable segments JoKey's cache memory remains free for other applications and may be used by them. Basically due to this feature the performance of JoKey rather rapidly becomes unreachable for other file systems (such as c-tree[9]) with growth of data amounts.

Data change safety. Without any user trouble JoKey avoids half changes of data files. It means both the possibility to renew data by concerted portions (transactions) and very small chances to cause a corruption of stored data by reason of hard- or software runtime failures.

Multitasking support. It is assured that many independent applications can work with the same table in parallel. The existence of a single copy of any table is achieved by the dual form of JoKey. The dual form implicates placement in a dynamic library of all the JoKey stuff except for low-level file operations detached into a special "file server" application.

It is noteworthy that the last feature affords the combining of the speed achievable by linking libraries and the safety and convenience of the client-server architecture. It is especially important for the artel being conspicuous by a plenitude of applications and the claimed possibility to combine them arbitrarily.

CONCLUSIONS

Let us reiterate the three main features of the proposed architecture.

1. The item phrased as a *fair division of control activities*. Parting of code and data resources is a common practice, meanwhile user interface and control functions usually remain in one hand. The artel breaks this concentration.
2. *The partition of applications* that comes even deeper than an "object-oriented" approach as it separates object features from objects themselves.
3. *The concept of links* (including virtual links) that covers a significant part of a system's functional filling.

Eventually all these properties contribute the system open character which does not require advocacy. However, it is the matter of the artel only (and the people who are using it). Still another artel feature having external effects should be stressed: *the AP-driven interaction that is very tight and intimate*. It means, for example, that traditional object modules have almost no flexibility advantages against artel executable modules. In order to use artel servants one must not obey the AP: it is enough just to utilize ductility of servants. There already are examples of employment of artel servants (e.g., the structure editor) in other systems professing quite different architecture principles.

ACKNOWLEDGMENT

We gratefully acknowledge the support of this work by the International Science Foundation (Grant RBQ300) and the Russian Basic Research Foundation (Grant 93-03-4727); we also highly appreciate fruitful colloquies with our colleagues Mikhail Koshelev and Andrey Kachalkov.

REFERENCES AND NOTES

(1) Sybyl software, version 6.1; Tripos Associates: Munich.
(2) Dyer, D. S. A dataflow toolkit for visualization. *IEEE Comput. Graph. Appl.* **1990**, *10* (*4*), 60−69.
(3) Upson, C.; Faulhaber, T.; Kamins, D.; Laidlaw, D.; Schlegel, D.; Vroom, J.; Gurwitz, R.; van Dam, A. The application visualization system: A computational environment for scientific visualization. *IEEE Comput. Graph. Appl.* **1989**, *9* (*4*), 30−42.
(4) Jacobson, A. S.; Berkin, A. L.; Orton, M. N. LinkWinds: Interactive Scientific Data Analysis and Visualization. *Comm. ACM* **1994**, *37* (*4*), 42−52.
(5) Gasteiger, J.; Ihlenfeldt, W. D. The WODCA System. In *Software Development in Chemistry 4*; Springer-Verlag: Berlin, 1990; pp 57−65.
(6) Ihlenfeldt, W.-D.; Takahashi, Y.; Abe, H.; Sasaki, S. Computation and Management of Chemical Properties in CACTVS: an Extensible Networked Approach toward Modularity and Compartibility. *J. Chem. Inf. Comput. Sci.* **1994**, *34*, 109−116.
(7) ChemArt itself is to be described in a separate paper.
(8) Brockschmidt, K. Inside OLE 2; Microsoft Press: Washington, 1994.
(9) c-tree Plus File Handler; FairCom Corporation: Columbia, 1990.

CI950180+