Doing Object Oriented Simulations: Advantages, New Development Tools

Jose M. Giron-Sierra and Juan A. Gomez-Pulido
Departamento de Informatica y Automatica
Facultad de Ciencias Fisicas
Universidad Complutense de Madrid
28040 Madrid, Spain

Abstract

In response to academic needs concerning modern digital control, we made a variety of educational aids based on simulation. We decided to employ OOP and selected some representative industrial subsystems to be simulated. OOP helped to analyze and implement the functions of the operational simulations, using an academic analogy for tasks distribution: lessons and teachers are "objects" coordinated to carry out the educational process, with the subsystems models as a resource. This distribution helps to structure the artificial intelligence, incorporated in the objects as specialized experts. We used Smalltalk for a rapid prototyping, and then C++ for the final versions. As the learning curve is slow for OOP, we began with a small team and small projects, building re-usable pieces of code, some of them for 3D animated graphics. We employed ObjectVision as a CASE tool to construct objects and generate C++ commented code. We used CommonView C++ classes, so the man-machine interface of our simulations is MS-Windows.

1. Object Oriented Technology

Along several generations, characterized by technological breakthroughs, the computers had been opening new potentials, to be substantiated by software. But this had never been easy. It is like having constantly new instruments, with new rules to be learned for its use. So the topic of software crisis ascends from old roots, and have several branches, sometimes prevalent.

Perhaps we are now in a similar situation, because hardware is much more powerful, and the users are now common people knowing what classical applications can do, so they want better things, with a lot of graphics and intuitive interfaces. It is possible, yes, to develop that kind of new applications: but with a big cost of time, investment, and long code. Software industry faces a difficult problem of project management to obtain finally, on time, a competitive sophisticated product with the new features wanted.

Object Oriented Programming (OOP) came with the promise of a new methodology, able to transmute the software design into the old human activity of choosing bricks, just as electronic design is now a matter of selecting integrated circuits (see [7] for a good book on OOP with many references; or [35] for theoretical concepts). In fact, some of the authors promoting this new concept [10] coined the term "software-IC", for speaking about classes and objects to software industry managers.

Many of the good ideas now in practice emanated from Xerox PARC, where the Software Concepts Group (SCG) was formed in 1970 to find and integrate two languages: one for functional description, to build the heart of applications, and other for a flexible, modern, human interface. The influence of the scandinavian concepts -Simula is a main representative, in actual use [21], with several descendants- was decisive. Through several versions, the SCG developed Smalltalk taking some ideas from Lisp, and adding to the classes and objects of Simula, the messages and the environment. From the beginning, the SCG considered that the object concept could be applied to the problem of interface development; and it has been demonstrated by the natural way Smalltalk includes windows, mouse, and graphics. It should be mentioned that the Xerox AI people started the line of object oriented dialects and evolutions of Lisp, sometimes employed for simulation, while some others from Xerox PARC were hired by Apple to develop some object oriented versions of Pascal; the tradition of object orientation at Apple was followed by...
MacApp and Hypercard. In the AI arena, it is clear the conceptual proximity of frames and objects, reflected by the characteristics of some of the modern tools for expert systems, which may include simulation capabilities [14].

It is important to note that object oriented languages were born for simulation purposes, as indicated by the name Simula. In particular, this language was conceived to model physical systems, to be used for nuclear engineering.

Whilst the scandinavian languages are compiled, thus making applications fast enough, Smalltalk is interpreted, slow, and you have to stay inside Smalltalk to run your application. But Smalltalk is good for graphics and windows. A look at the published object oriented simulations (the number of conferences and reunions related with this topic is exponentially increasing; see [15] for specific references, or [37] for a modern perspective on intelligent DEVS), reveals a majority of discrete-event applications, in Smalltalk. The reason could be the importance now given to manufacture; but is also true that Smalltalk adapts very well to simple animated graphics and to discrete and not fast event handling. Presently the situation is changing, because there are versions of Smalltalk for compiling, and also C++ offers an interesting alternative, open to the use of different libraries.

Simulation is not the only driving force to promote OOP. We suspect it is more important, so new tools are emerging, the credit given by software industry to this technology, as a mean to progress and not die. A profile of time investment to develop an application shows tremendous amounts dedicated to modern iconic interfaces, and to the final step of integrating, testing, debugging. The proportion of code devoted to graphical interfaces (GUIs) is more and more significant. Considering that OOP is well suited for windows and mouse (discrete event, forms including variables and behaviors), it is a good decision to adopt that methodology for the interface. Typically this decision entrains a consequence: developing the rest of the application code with the same object oriented methodology. One of the advantages is that you enter a modular way of programming, easy to distribute and coordinate among team members, with possible code re-use for other products.

Relational data bases look at reality the same way a grammarian analyzes sentences, finding the nouns to which predicates are attached. To design a relational data base you have to discover which are the basic entities, to assign attributes (pointers, other files). This way of looking at the problem is close to the object oriented style, which, in addition, extends the relational concept by attaching procedures to classes. This proximity offers a smooth passage to achieve data base applications, with the new object oriented technology.

In agreement with our comments, it is observed that the number of commercial tools for developing object oriented applications is now growing, with emphasis on data bases and windows. We have used some of these tools, and we shall detail our experiences in this paper.

As the new methodology is carried to real implantation, people is realizing that it implies a change in several dimensions. First, programmers have to think in a new way, and it takes time and humour. It is not a simple, but long, learning curve. You have to alter your analyzing habits and your working scheme. Second, the project management standards must be changed, because the classical top-down approach is not more the absolute paradigm [16]. As noted by B. Meyer [26], it is a new culture, with recursive strategies starting down- up from simple components and first prototypes to be enriched. It is not a surprise that now, one of the main activities around OOP, is learning, training for.

2. Our Research

Our Department of Computer Science and Automatic Control, is constantly adapting to a rapidly changing scene of scientific contents and social needs. There is an increasing application of modern control ideas in the industry, as the digital technology offers a competitive, more powerful, solution for classical and new problems. This implies the need of people educated in the theoretical [19], and practical, basis of the industrial digital control use. In our academic response to that need, the most challenging aspect is the practical knowledge: the students require some intuitive sense of what the controlled processes are, and what the control algorithms can do. One of the most important decisions we took, was to establish a new Laboratory for Automatic Control [4], where we are including some small scale components of industrial processes, interacting with personal computers. Some of the experimental devices are now, a two-tank system, a heat exchanger, an A.C. motor coupled to a D.C. generator, and a distillation system.

Soon we discovered that these experimental practices could be dangerous for the students and for the equipment. So we decided to develop some training simulations on computer, to prepare the students before coming to real practices. That was the starting point of our research.
Attracted by the OOP potentials, we decided to use Smalltalk/V [11] (good for personal computers) to develop the training systems. We began by simulating the action of the experimental devices, which are continuous time processes. In a natural way, we considered the components of the devices (tanks, valves, motors, etc.) as objects, with their pertinent variables and behaviors. Each of the objects include some artificial intelligence features, to give advice for normal and difficult situations ([27],[28],[13],[34]).

Then we extended the object oriented analysis to the educational activity of the training systems ([25],[20],[36]). Just as the knowledge engineers capture the role of human protagonists, we looked at the academic organization surrounding the laboratory, and induced the specific contribution of each person. In the training systems there are objects corresponding to laboratory assistants, the supervisor, the teacher of theory, etc. These objects use for the training some academic resources, such the simulations of the practices, and interact via a blackboard.

As the programs gained in complexity, Smalltalk/V showed to be slow for our purposes, so we shifted to C++.

Later on, as we developed interesting software modules, we started to re-use these modules to build some educational units. The units are conceived for the students to use for personal experiments. Some of the units are a personalized version of the training systems, adding a tutorial. Others follow the standard block diagram approach for general study; and, finally, others are important examples found in the textbooks. In all cases, we employed object-oriented simulations for the processes and the educational organization, and distributed artificial intelligence features. We applied some modern tools for CASE, to obtain well structured C++ code and to adhere to the MS-Windows standard interface.

3. Using Smalltalk

Our first use of Smalltalk demonstrated how good is for rapid prototyping, with the advantages of an integrated environment having browsers and debugger.

Learning to use Smalltalk has three phases [21]. In the beginning some time must be allowed for mind changing, to assimilate object oriented concepts; then it is easy to create first prototypes, with graphic impact, and see what happen. But the third phase is harder, because you decide to read the enormous reference section of the manual and use advanced features. In particular, the task of developing human interfaces is complicated.

The natural way of simulating processes in OOP, is to see each component of a process as an object, and let the objects interact according to the structure of the process (this could lead to asynchronous interaction). This scheme is easy to implement for discrete-event processes; but there is a problem with simultaneous events, from the very beginning of simulation science, as we use Von Neumann computers with only one processor acting in a sequential manner. Parallel computers do not address this problem. Object oriented simulations could be achieved with several processors, one for each of the objects; but it is hard to conceive a general purpose architecture of economical interest. There are several software alternatives to handle concurrency and multiprocessing [8]. For instance, the Goodies 1 Application Pack [12] includes some classes to manage multiprocessing by semaphores (as the manual says, is especially useful in discrete-event simulation).

One of the solutions for complicated cases is to co-ordinate the interactions, creating an object with supervisory and governing functions over the component simulating objects. This new object corresponds to the process structure, and, in some sense, reflects the "gestalt" intuition: an entity is more that the simple aggregation of its components. In our case, the processes are continuous time, and that kind of solution is appropriate: we chose a sampling period, short enough to follow the process dynamics, and the "process" object achieves the interactions in the sampling instants, sending and receiving messages. Between these instants, the simulation objects calculate their next messages (perhaps integrating differential equations) to the "process" object.

Suppose a two tank system, at different level, where a valve could interrupt the flow of water from the bottom of first tank, to the second tank. If you close the valve, it sends a message to the "process" object, which, in turn, send
messages to the two tanks informing of the new situation for level calculations (Figure 1). The "process" object has a causal diagram of the process, to orientate normal scheduling and mediation. The diagram is also employed by the artificial intelligence features of the "process" object, to simulate faults and help to diagnose the problem.

The simple inference engines embodied in the objects are codified in Smalltalk language.

Our experience is that hierarchies must be simple, few levels: include more features in the objects. One of the main reasons for slow performances, is having Smalltalk searching for methods across several levels of a hierarchy.

It is better to organize the classes in several clusters, one for physical modelling, other for graphical expression of behaviors, other for windows and menus. Each cluster with its own hierarchy.

The software scene is continuously changing. When we started to use Smalltalk/V, we had no way of implementing autonomous applications as executable files. The Smalltalk-80 alternative, once confined to mainframes, offered no color. To enrich the library of classes it was possible only to buy some cheap sets (we mentioned above a member of the Goodies series).

Now there is a new situation. Recently has been announced a version of Smalltalk-80 [31], in color, portable to MS-Windows 3, Macintosh, OSF/Motif (X-Windows), and Open Look (Sun), bringing executable applications. Also, there are several new sets of classes, such Widgets/V [1] for a much better interface, or a complete multiprocessing kit, or a simulation extensions kit. The vendors suppose you are using a good, fast, computer. If not, there are some accelerating boards for PCs, specially designed for Smalltalk. We are thinking about coming back to this paradigmatic environment.

4. Using C++

When entering the C++ domain, you get more freedom but not ready made integrated solutions. Smalltalk helps you to edit, debug, test, without exiting the environment; you can use the windows of Smalltalk as the windows of your application; and you enjoy a universe of classes for different processing and displaying needs. With C++ you have to get (buy) external libraries for non-rudimentary interfaces and processing; and get editors, compilers, utilities. This could be seen as an advantage, or a difficulty. Recently, Turbo C++ [6] appeared with the typical Borland friendly environment, including editor and debugger.

The transition from Smalltalk to C++ meant for us a new territory to explore, with evident risks: the proximity of C, so open to comply with your personality, is a temptation for wild programmers to work out almost cryptic code. In fact, with C++ you can exercise several programming alternatives: classical, structured, object oriented, hybrids. We looked for a software tool to guarantee the development of good, structured, object oriented C++ code. Recently a CASE environment came into light, satisfying our requirements. The name of that package, ObjectVision [29], marks the main feature: visual programming of objects.

With ObjectVision you build object oriented applications "drawing" on screen your program's hierarchy of objects; then, ObjectVision converts your diagram into commented, ready to compile code in C++ or Turbo Pascal 5.5. Suppose you want a hierarchy of several objects; there is a menu with several icons to choose; with the mouse you select the "object" icon and, clicking the mouse, you put the top object on screen; then, by selecting and using other icons, you attach properties and methods to the object; with the mouse you put new empty objects on screen, and drawing lines you connect them to the top object: the objects inherit the properties and methods of the top object; you can add other properties and methods to the new objects. Your program is functional as you draw it; as fast as you draw objects, you can test them out to see how they work.

You can ask ObjectVision to generate only classes, or the full commented code. The visual programming environment includes windows, icons for discrete-event simulation, and importing functions for bitmap graphics. You can use these interface elements for your own application.

We applied ObjectVision to prototype the code, one of the best uses of this tool. Then we employed a professional editor for programs [32] to tailor the results.

As a general rule to guide our project management, we concentrate the efforts first on modelling and simulation (equations, structure, interactions, time), then we included graphics, and finally the human interface. ObjectVision served for the first step.

Some of the graphics are designed for the tutorial and helping functions. It takes time and inspiration to accomplish images of pedagogic value; so we decided to establish bridges to the modern software tools for personal publishing. We created C++ functions able to import CAD and Clip-Art files, in such a way that our applications can put these images on screen in a flexible way (the desired size, position, color, aspect).
We employed DesignCAD 3D [2] for some complicated pictures. Once created a 3D graphical object, DesignCAD 3D can save onto files several different 2D views of the same object. This is useful for some descriptive steps of the tutorials.

By means of Art & Letters [9], Harvard Graphics [33], Applause II [3], and other programs handling clip art, we created some CGM files with conventional graphs and standard images.

With the help of simple spreadsheets and text processors, we stored onto files data blocks (for instance, steam tables) and the paragraphs for help and tutorials. Some C++ functions were created to import these files.

The functions for file importing and displaying the contents, are included as methods of an object called "artist".

For the human interface, we wish to adhere to the most widely-used standards. The mouse driven iconic GUI of the Macintosh is the first and very successful example of such an standard. In our case, we employ Intel based computers, and the situation about GUIs is not so clear: but MS-Windows seems to be a promising standard for the next future, gaining rapidly adepts, these days, with the version Windows 3.

So we looked for an object oriented software tool, to ease the engagement with MS-Windows. A first look to the MS-Windows System Development Kit revealed a very difficult access. But recently the CommonView SDK [18], appeared with many good features for our purposes. This package is an extensible object oriented framework built with C++, providing a class library which represents typical GUI elements like windows, menus and other accessories; specially designed for portability, so you can use the same source code to develop applications for MS-Windows, Presentation Manager, HP New Wave, Macintosh, or OSF/Motif and X-Windows.

Using CommonView, which includes the famed Glockenspiel C++ compiler, was easy to integrate MS-Windows as the human interface for our applications. Some simple functions of MS-Windows were used as primitives by the artist object, to display the images, and by the simulations for visualizing process dynamics.

Taking advantage of the CommonView features, we plan to port our applications to the Unix and X-Windows sphere. That means a better use of modern hardware, in the form of workstations, X-terminals, and shared resources.

5. The Groups

Along some years of work, we had several teams of students and collaborators. The general problem has been to assure the learning, on time, of the required concepts and techniques.

We employed Turbo-C [5] as a first step to familiarize with good programming practices. The integrated environment of this package, is an effective aid for obtaining good results in a short time (this is an important factor to inject optimism to novels).

Our philosophy has been to conduct the research, and the associated learning of tools and techniques, along short-term subprojects (success is motivating), with publishable results. Students want spectacular effects, with colourful windows and impressive graphics, but we want to have before the substance: good modeling and simulation, useful scientific features. It required some discipline, but we
came straight to obtaining correct equations with Turbo-C. Then we prepared a stock of files with images for the tutorial and advising functions, and learnt to use scientific libraries for results visualization. In this way we obtained first versions of educational instruments: a good reference for the next steps. (Figure 3).

The transition to OOP was initiated with Smalltalk/V; examining the materials already developed in a previous work, and doing some prototypes about the new laboratory devices [23]. Then we began to use ObjectVision to accomplish the modeling and simulation in C++. With the help of Turbo-C++ the code obtained with ObjectVision was refined, and interactively tested. We obtained a second version of the educational instruments, in object oriented form. That meant the development of many interesting re-usable pieces of code.

When the regular academic activity opened the laboratories, to carry out the experimental practices, we effectuated some knowledge acquisition activities. The objective was to induce rules for the artificial intelligence embodied in our educational instruments. We applied techniques from the psychological theory of groups, to extract some rules about students in laboratory: it required intensive independent observation, and some questionnaires. The rules underneath the laboratory assistants behavior, are crucial to orientate the activity of the training systems: for instance, the measures adopted to avoid danger.

To systematize all the construction and testing operations about knowledge bases, we employed VP-Expert [30]. There are many other alternatives, but this one is adequate for the low level of complexity we want to keep [22]. Again, with this subproblem we obtained some final results, of educational science interest.

When the students start to connect instruments, motors, valves, etc., they suffer serious difficulties and errors, because they are not familiarized with real things. We remarked that seeing is not understanding. This conforms to the recent criticisms about animated graphics simulation [17], concluding that it is of limited utility (abstract tools and figures are better for engineers). Our conclusion for tutorials and advice, is that some text and scientific graphics is required. The students must see and think. Analyzing the comments of laboratory assistants before the practices, we concluded that a component of their expert knowledge is a sense of the orders of magnitude, the relative proportions, and simple object-attached rules: for instance those concerning the correct handling and the possible difficulties with multimeters. One of the main educational results of laboratories is the vital assimilation of this knowledge by the students.

Using the conclusions relative to expert rules, the following step was to distribute intelligence among the objects constituting our educational instruments. Some of these objects are motors, tanks, measuring devices. Others mimic the activity of laboratory assistants.

As a final step, we employed CommonView for including MS-Windows as the interface. It is important to go slowly, from simple menus to refined ad-hoc structures of child windows.

We accompanied the programming activity with short seminars, to introduce the main concepts and techniques needed. About simulation, C, OOP and Smalltalk, CAD and publishing tools, C++ and CASE, expert systems, human interfaces.
6. Conclusions And Perspectives

We have found a set of tools to develop object oriented, intelligent, graphical applications in C++. The need of these tools is a hot topic, and the software industry is reacting with new products. For instance, it was recently announced a new version of ObjectVision, which adhere to MS-Windows.

The object oriented approach is adequate for simulation, with important benefits for the software and project management: software re-use, modular and structured, divide-and-conquer strategy. But it requires learning. Our experience is favourable to learning by doing, through small projects or subproblems.

It is interesting to note the good results derived from our educational metaphor: objects corresponding to different roles (teacher, laboratory assistant, artist...). This analogy helps to look at the problem in hand, training and personal experimentation, and have classical references to isolate and assign tasks. In our case, even the laboratory guides are good sources of knowledge to know what to implement. In the industrial cases, the sources could be the operations manuals, and the procedures, persons and activities dedicated to training; we think that dissemination of high technology will mean more learning and training needs.

For the next future we have a hard work of improving and completing our simulation applications, adding new features and more experimental practices. We plan to port the applications to Unix and X-Windows, to abandon the MS-DOS limitations and go for industrial more powerful applications.

7. References


