Object-Oriented Programming for Project Management Software

Thomas M. Froese, Research Assistant
Boyd C. Paulson, Jr., Ohbayashi Professor of Engineering

Dept. of Civil Engineering, Stanford University
Stanford, CA, 94305-4020, U.S.A.

ABSTRACT

What does object-oriented programming have to offer project management software systems? This question is examined by first seeking out those development trends and directions that will be of importance for project management systems in the future. These are found to be flexibility, integration and intelligence. The basic characteristics of object-oriented programming are then described; namely the use of objects and encapsulation, messaging and polymorphism, hierarchies and inheritance, and a new overall approach to programming. Three components of object-oriented application design are also discussed: the model, the mechanics, and the conventions. Finally, the advantages that the basic object-oriented characteristics and design approach offer for moving project management systems (and architecture-engineering-construction management and automation systems in general) towards the identified goals are presented.

1. INTRODUCTION

In the future, project management systems will be more flexible, will be highly integrated, and will be significantly more intelligent. This paper argues that the principles of object-oriented programming will make significant contributions to the advancement of project management software in each of these directions. The paper will first analyze these expected future directions of project management software. The basic characteristics of object-orient programming are reviewed. An overall approach to the design and development of large object-oriented systems is then presented. Finally, we show how this course of development would, in fact, support the software trends that we have identified as being significant for project management systems. While our focus in this paper is on project management software, we believe that the results are equally applicable to other future construction management, automation, and robotics software.

2. PROJECT MANAGEMENT SYSTEMS: CURRENT STATE AND FUTURE DIRECTIONS

Prior to the advent of network techniques and "modern project management" (in the late 1950's [Fondahl 87]), construction planning and scheduling was based solely on the intuition and expertise of experienced site personnel. This "first generation" [Lichtenberg 74] of project planning was superseded by a second generation of network-based, computer-assisted techniques. As these computer systems grew larger and more sophisticated, they also became more opaque and dictatorial, pushing the more competent human planner out of the process. A general dissatisfaction with these inflexible computer systems led to the birth of a third generation of project planning, most notably characterized by the re-ascension of the human planner to the central role using highly interactive methodologies. In the early 1970's, this third generation first appeared in research institutions [Paulson 72, 73], [Lichtenberg 74]; in the late 1970's it was implemented in large-scale systems [Paulson 75]; and in the 1980's it formed the foundation of an explosion in micro-computer based project planning systems [Fersko-Weiss 89]. Although

monumental advances have been made in personal computing hardware and graphical interface software, most current project management systems have evolved little in terms of basic approaches to planning, underlying algorithms, and levels of system integration over the past 15 years.

> With the exception of a limited number of features such as resource loading, hardware compatibility, enhanced graphics, and the vendors' perception of "user friendliness," the programs are functionally very much alike and have not changed dramatically since the introduction of PERT/Cost ([Badger et al 87], p. 100).

In spite of the underlying uniformity of commercially available project management systems over the past decade, the continued advancement of computing technology along with the pervasive general dissatisfaction with project planning systems and the level of research activity in this field have led many people to believe that project management systems will be quite different in the near future. Several recuring themes appear among published forecasts of project management systems to come [Ibbs 85], [Thisner et al 87], [Hansen 87], [Levitt 87], [Logcher 87], [Badger et al 87], [Ibbs et al 87], [Teicholz 87]. These themes are summarized by flexibility, integration and intelligence.

2.1. Flexibility

Whereas rigid systems execute predetermined sequences of tasks in immutable fashion, flexible systems are highly configurable and can perform their duties in many different ways and in indeterminate procession. We believe that flexibility is particularly important for project management systems. This is because project management, by definition, involves a dynamic rather than a static domain. The characteristics of most projects change constantly and rapidly. Project management is performed by individuals or small groups who inject a high degree of personal style into the process. These working styles differ dramatically from one team to another, and they change during the life of the project. As a result, the nature of the output required of a project management software tool, the level detail desired, the amount of time and effort that can be afforded to the system, and many other characteristics of the system's environment vary frequently. High system flexibility is required in this dynamic setting.

2.2. Integration

Project management involves many different areas of concern, each of which deals with the same project and the same body of project-related information. To an increasing extent, the software tools used to support these various tasks will become integrated into cohesive overall systems. There are numerous facets to this integration. *Functional integration* deals with the offering of many different functional capabilities by a single unifying system. The various project management tasks and software tools that support them are, in fact, highly interrelated; functional integration allows the user to jump back and forth between these tasks. Functional integration also allows the user to interact with a single uniform interface to the assorted subsystems. *Data integration* focuses on the sharing of data and information among the many software tasks involved in an overall integrated system. Data sharing and connectivity greatly reduce the input requirements for the individual pieces of the system, since the data generated by one software component can be maintained and used by another. Enabling many subsystems to share the same pool of information also provides greater data consistency, and simplifies programming by allowing all data to be treated in a uniform manner. Finally, *technological integration* relates to the envelopment of many different software technologies within one overall system. The collaborative use of

traditional structured programming, databases, and knowledge-based processing components is an example of technological integration.

2.3. Intelligence and Information Management

A common theme among the references reviewed is that future project management systems must provide more information and less data. It is generally expected that this capability is best provided through the use of knowledge-based and artificial intelligence capabilities. System intelligence is the key contributor to better and more meaningful system output. Intelligence also allows reduced input by allowing necessary information to be derived rather than entered.

2.4. Combined Directions

> Our future systems will be far more integrated and deal in a common manner with all types of project, environment, organizational and other types of data, using knowledge to integrate and utilize the data ([Logcher 87], p.86).

It is interesting to note that these expectations of future systems are very similar to the expectations which prompted the development of third generation systems in the 1970's [Paulson 72]. Much of the conceptual framework developed then is still valid today, but the advances in computer technologies have given the manifestation of this framework a whole new meaning. Within the current technological context, much work has already been done in each of these three areas. Commercial project management software packages continually strive to become more flexible, particularly through better interfaces and greater configurability. Levels of system integration are also increasing both in commercial and in research systems. This integration takes the form of extended functionality added to existing systems, file exchange capabilities between software packages, and the increasing use of project-wide databases. The application of AI to project management technology has perhaps been the most active area of computer-aided project management research for many years.

To a large extent, however, these efforts have taken place independently of each other. An example of this phenomenon can be seen in recent efforts to apply AI to project management systems. Worldwide research in this area has led to significant advances in adding "intelligence" to systems. However, we believe that AI and expert systems will not provide practical overall tools by themselves, but rather will provide key components of larger hybrid systems. This integration of AI systems into general hybrid environments has been excluded from the scope of the majority of AI research projects. In object-oriented programming, we see not only the potential to support advances in each of the three future trend areas described, but also the ability to provide a unifying framework for work in these directions and for linkages between project management systems and other fields such as construction automation and robotics.

3. CHARACTERISTICS OF OBJECT-ORIENTED PROGRAMMING

The preceding section has outlined a set of goals for future project management systems. This paper suggests that an object-oriented approach will contribute to the attainment of these goals. This section outlines the basic characteristics of object-oriented paradigms that offer particular promise for supporting the stated system goals. Section 4 discusses the application of these principles to overall system development.

One of the fundamental attributes of object-oriented paradigms is use of objects and encapsulation. The underlying idea is that importance lies in *what* operations can be applied to data, not in *how* these operations are performed or how the data is represented. An abstract data-type is therefore defined in which the description of its operations is public, but the representation and implementation details are kept private. This data-type is called an *object*, and its operations are called *methods*. Encapsulation strengthens the concept of objects further by stating that an object's data *cannot* be accessed directly: all interaction must be at the object level through its methods.

This interaction with objects is performed by sending *messages* to objects that trigger their methods. Messages can be used to provide data to an object, to request that an object return data, or to instruct an object to perform some action. Typically the sender of the message is only concerned with an object's response and is neither aware nor interested in the exact mechanisms triggered by the message. In fact, different objects can respond to the same message in very different ways. This characteristic, called *polymorphism*, allows diverse types of objects to all be treated in a very uniform manner.

Objects are normally organized into *hierarchies* such that each object represents a subclass or "type of" another object. This allows objects to *inherit* capabilities from other objects, a feature which provides many efficiencies.

Finally, object-oriented programming involves a different way of thinking for program developers. The conceptualization of a program as object representations of domain components, rather than as data structures and control flows, leads to many of the specific advantages that will be discussed in section 5.

4. AN OBJECT-ORIENTED APPROACH

Although the concepts that define object-oriented programming are fairly well established, their application to actual programs has been carried out to various extents in many different forms. Examples of these diverse applications include the reimplementation of traditional programs using object-oriented versions of standard programming languages (such as C++), the development of AI programs using frame-based representations, or the addition of object-oriented interfaces to relational databases. However, our interest lies in a comprehensive application of object-oriented principles to the field of project management systems as a whole, requiring a thorough and novel software design effort. We see three main components to this design: the model, the mechanics, and the conventions. While there is insufficient room in this paper to examine a specific object-oriented design for a project management system or even to describe the design process in detail, we will discuss these three main design components and identify some of the key issues that arise.

4.1. The Object-Oriented Model

The central step in the development of a large object-oriented system is to develop a model that describes all the pieces of the system in terms of objects. Specifically, the model identifies the name and purpose of each object in the system and defines the objects' basic capabilities by describing their methods. In addition, the model should provide some organization to the system's objects by categorizing and grouping them according to one or more points of view. The model should also describe the major interactions between objects in the system (i.e., which objects send what types of messages to which other objects). Finally, the model should deal with any relevant system-level—as opposed to individual object-level—issues (examples of system-level issues are discussed in section

4.3). Once constructed, this object-oriented model serves as the common structure around which all future development, implementation and maintenance is performed.

The creation of the model is not a routine matter. There are clearly many different ways in which a large system can be divided into individual objects, and while no one partitioning is correct, some will be preferable to others. The best guide for determining what the objects should be is to study the real-world domain to which the system applies. The individual objects within the systems should, to a high degree, mimic the individual components found in the domain. In some cases this will lead to objects that correspond to specific physical components. Objects could be created, for example, to represent the beams, columns, walls, and other physical pieces of a constructed facility. At other times, however, aspects of a domain can be better divided into more abstract components or processes. For example, in a project management system it may be useful to define objects that symbolize a plan or that represent the estimating process.

This mimicking of the domain's natural partitions provides a starting point for defining the model's objects. However, several other considerations also influence the selection of objects. First, inheritance hierarchies should be exploited, allowing proprieties and methods that are shared by more than one object to reside with a common parent object. Second, objects are typically referred to or *owned by* a single object that has primary responsibility for its creation, persistence, and destruction. Thus objects with similar ownership can be organized into ownership groupings. Third, aggregation hierarchies defining those objects that are *parts of* more general objects (such as a beam object being a part of a structural frame object or an activity object being a part of a network object) can be used to allow information to be summarized or refined to varying levels of granularity.

4.2. Mechanics

Whereas modelling deals with the mapping of object-oriented programming principles to some problem domain, mechanics deals with the issue of how these principles of encapsulation, messaging, and inheritance are actually implemented in a computer environment. Fortunately, the many object-oriented programming languages and environments which now exist (e.g., SmallTalk, C++, object-oriented LISP languages like CLOS or CLOOPS, or AI environments like KEE) provide these basic mechanisms, and often provide many more features such as predefined user interface objects or object archiving. Nevertheless, the software mechanics provided by these languages may not be sufficient for specific applications, and additional constructs or restrictions may be warranted. For example, it may be desirable to utilize more than one language in the implementation of a large system, such as C++ for algorithmic calculations and CLOS for knowledge-based reasoning components. The development of procedures for sending messages between objects that are defined in different languages is an example of the software mechanics issues that must be considered as part of the design of a large object-oriented system.

4.3. Conventions

One of the objectives of object-oriented paradigms is that all objects can be treated in a similar manner. It is also intended that a programmer can interact with an object while having only minimal information about it. These objectives require a high degree of uniformity in the specification of each object's methods, yet there is nothing to enforce this within the modelling process or the basic mechanisms. Rather, this uniformity must come through design and programming conventions. As a simple example, Objective-C adopts the convention that messages return their object's identifier when no other specific return value is warranted. This allows multiple messages sent to a single object to be nested, as

follows (where *activity_object* is an object representing an activity in a CPM network and *setDuration:*, *update_network*, and *get_float* are messages that it understands):

```
[[[activity_object setDuration: 5] update_network] get_float];
```

Well-defined conventions also provide solutions to system-level problems within an object-oriented application. For example, it may seem appropriate for an object representing a CPM activity to message an activity-inspector object (a form of user interface window) if its duration has been changed so that the inspector can update its duration display. However, a change in the inspector's duration display requires that the activity object be messaged to change its duration, since the change may be initiated by the user. Each message seems appropriate by itself, but together they form an infinite loop. A solution to this problem may be to propagate newly assigned values only if they differ from previously stored values, or if the change was self-initiated. Of significance is that the solution to this particular problem can lead to the solution of many similar problems if it is imposed as a programming convention on all objects.

Finally, in order to allow programmers to efficiently utilize the objects that have been developed, it is important to adopt conventions that standardize the documentation of each object, as well as to document the conventions and standards.

5. CONTRIBUTIONS OF OBJECT-ORIENTED PROGRAMMING TO FUTURE PROJECT MANAGEMENT SYSTEMS

We have discussed the basic characteristics of object-oriented programming and outlined an approach to object-oriented design. Many benefits are traditionally attributed to object-oriented programming, such as rapid prototyping and decreased development time, a high degree of code reusability, and improved software maintenance. These benefits will all apply to project management programs as well. In this section, however, we will describe the benefits available to project management systems as they relate specifically to the developmental objectives that were presented in section 2.

5.1. Flexibility

Object-oriented programming lends itself well to flexible systems. Configurablility and customization is supported by the high degree of modularity and the interface/ implementation dichotomy. Objects can be altered drastically, yet if they conform to the same interface specifications, the alterations are irrelevant to all other parts of the system. Furthermore, the ability to treat very different objects in a similar manner allows for flexibility in implementation, since it enables seamless integration of traditional procedural components, data bases, AI modules, and many other diverse techniques. Finally, the paradigm supports varied and arbitrarily-sequenced processing. The dominance of a central flow of processing control is diminished. Encapsulation helps to ensure that messages can be passed to objects in any order, since the object itself can check for unsatisfied pre-conditions before data is accessed, and can consequently return appropriate warnings or perform the preliminary requirements. Using all of these capabilities, the flexibility of large integrated systems will be significantly enhanced.

5.2. Integration

The object-oriented approach offers several distinct advantages for developing integrated software systems. Among these are global orientation, public object interfaces, private object implementations, and implementation flexibility.

First, the object-oriented model is structured predominantly around the natural organization of the problem domain, not around some specific programming task. Similarly the design of the individual objects parallels the inherent functionalities of the domain components or processes. As a result of this focus on the domain as a whole rather than on a specific programming task, both the overall architecture and the detailed object designs tend to be equally appropriate to most other domain-related tasks (i.e., they are appropriate for functionally integrated systems).

Second, the interface of each object is public and well-defined. Because of this, all objects in a large system can easily interact with each other (for example, an object relating to an estimating module can access the duration of an activity object in the scheduling module as easily as an adjacent activity object could). Yet the modularity and the formality of the interface definitions aids in the organization and management of large systems.

Third, the implementation of objects is private. Not only does this avail the objects' users of the need to know the implementation details, but the encapsulation means that these details won't adversely interact and conflict with other objects. This allows for the graceful growth of large systems. Encapsulation also allows tight data management by ensuring, for example, that inquirers have permission to access the data, that data is entered or modified in a consistent manner, and that other system components are notified of changes as required.

Fourth, a variety of implementation technologies can be used as long as they support the basic object-oriented interface (again because of private implementations). For example, objects can be implemented using standard structured programming techniques, database methodologies, or knowledge-based processing. The choice could be fully independent of the public interface since polymorphism allows all of these implementations to respond in a uniform manner.

5.3. Intelligence

Greater levels of intelligence will come to project management systems through the field of artificial intelligence, and many researchers are currently working in this area. The majority of current AI systems adopt a frame-based or other object-oriented representation approach. If fact, the object-oriented modelling process described above is quite similar to domain modelling processes used in AI. This provides the opportunity for significant cross-fertilization in the modelling of project management endeavors between AI and general object-oriented approaches.

More significantly, perhaps, object-oriented approaches support the development of hybrid AI, database, and traditional project management systems. This support comes from the ability to combine multiple implementation technologies (technological integration, discussed in section 5.1) and through the uniform domain modelling approach. The implications of incorporating AI include the need to support mechanisms for symbolic data types and processing.

6. CONCLUSION

Having become familiar with object-oriented techniques and having used them in several AI applications for construction robotics and management, we were left with the feeling that they offered many advantages for our planned future software development efforts. In this paper, we have attempted to explore this intuition and to justify or refute its validity. We have found that there are indeed many specific reasons why object-oriented programming is well suited to the types of project management software that we believe

will be developed in the near future. We can also report that these advantages have, by and large, been borne out in our application development efforts to date. We intend to continue to work in this area, which we believe will lead to issues of what the fundamental pieces or building blocks of the construction effort are, and of how these pieces interact. We see these issues as being central to much other work in construction management and automation.

BIBLIOGRAPHY

Badger, A.A., Reinschmidt, K.A. and Gandt, A.R., "Project Control System Integration," *Project Controls: Needs and Solutions,* Ibbs, C. William, Jr. and Ashley, David B. Eds. New York: American Society for Civil Engineers, 1987, pp. 88-100.

Fersko-Weiss, Henry, "One Project, 3,000 Tasks: High-end Project Managers Make the Plans," *PC Magazine*, Vol. 8, No. 9, May 1989. pp. 155-195, 1987.

Fondahl, John W., "The History of Modern Project Management, Precedence Diagraming Methods: Origins and Early Development." *Project Management Journal*, Vol.18, No. 2, June 1987. pp. 33-36.

Hansen, Soren, "Hardware & Software Implications on: PM and the Computer: The Year 2001," *Project Management Journal*, Vol.18, No. 3, August 1987. pp. 47-48.

Ibbs, C. William, Jr., *Proceedings of a Workshop for the Development of New Research Directions in Computerized Applications to Construction Engineering and Management Studies,* Urbana: University of Illinois at Urbana-Champaign, Dept. of Civil Eng., 1985.

Ibbs, C. William, Jr., Ashley, David B, Neil, James M. and Feiler, Frank W., "An Implementation Strategy for Improving Project Control Systems" *Project Controls: Needs and Solutions,* Ibbs, C. William, Jr. and Ashley, David B. Eds. New York: American Society for Civil Engineers, 1987, pp. 101-112.

Levitt, Raymond E., "Expert Systems in Construction: State of the Art," in *Expert systems for Civil Engineers: Technology and Application,* Maher, Mary L. ed., New York: The American Society for Civil Engineers, 1987, pp. 85-112.

Lichtenberg, Steen, *Project Planning - a Third Generation Approach*, Polyteknisk Forlag:Copenhagen 1974.

Logcher, Robert D., "Adding Knowledge Based Systems Technology to Project Control Systems," *Project Controls: Needs and Solutions,* Ibbs, C. William, Jr. and Ashley, David B. Eds. New York: American Society for Civil Engineers, 1987, pp. 76-87.

Paulson, Boyd C., Jr., "Man-Computer Concepts for Planning and Scheduling," *Journal of the Construction Division*, ASCE, Vol. 98, No. CO2, September, 1972, pp. 275-286.

Paulson, Boyd C., Jr., "Project Planning and Scheduling: Unified Approach," *Journal of the Construction Division*, ASCE, Vol. 99, No. CO1, July 1973, pp. 45-58.

Paulson, Boyd C., Jr., *Continuing research in the Development of Interactive Man-computer Systems for Engineering-construction Projects,* Technical Report No. 200, The Construction Institute, Department of Civil Engineering, Stanford University, CA, September 1975.

Teicholz, Paul M., "Current Needs for Cost Control Systems," *Project Controls: Needs and Solutions,* Ibbs, C. William, Jr. and Ashley, David B. Eds. New York: American Society for Civil Engineers, 1987, pp. 47-57.

Thisner, Anders, Teicholz, Paul M. and Havas, George, "PM and the Computer: The Year 2001," *Project Management Journal*, Vol.18, No. 3, August 1987, pp. 39-45