The Viability of the Object Paradigm for Shared Database Applications: A Case Study

Frederick G. Kohun, Robert J. Skovira, David F. Wood
Robert Morris College, Pittsburgh, PA 15219

ABSTRACT

While a predominant portion of the extensive literature on object programming and technology addresses conceptual foundations and object terminology, there has been little work on the issues of non-systems related applications development. Current object research has lent itself to the issues of software reusability, with a primary focus on tools directed toward graphics and graphic user interfaces (GUI). This paper examines the problems and possible solutions of applying the traditional object paradigm to the design and development of a college student housing selection/allocation and management system by utilizing M.

INTRODUCTION

Over the past twenty-five years the literature on object oriented technology has presented and discussed both old and new terminology without the benefit of significant or radical contributions to the overall object paradigm. The name change from object oriented programming to the generic object oriented technology is the result of the expansion of the object paradigm to include object oriented analysis and design and object oriented data bases.

The object paradigm is a perspective on software engineering. It differs from the *functional* view that procedures with data structures are the constituents of software systems[14]. This view sees parts of a program as separate entities called *objects*. These program components or *objects* model phenomena by tying together in a bundle the data attributes, operations or methods, and default state[12],[13] of an individual thing, event, or situation. The paradigm also theorizes about *classification* by which is meant data abstraction as classes, *polymorphism* which concerns reuse of methods across classes of objects, and *inheritance* which refers to the cloning of attributes and methods thus creating subclasses of objects[12].

The literature does in fact demonstrate that there is enough of a disparity over the nature of the object paradigm that, although virtually every professional recognizes its advantages, and that vendors will eventually support it, no one agrees on how to use it to design, develop and implement a typical business application [1], [2]. Figure 1 juxtaposes the disparity in the conceptual basis for the object paradigm.

Figure 1: Sample of Object Oriented Technology Definitions

	Booch [3]	Coad & Yourdon [5]	Page-Jones [6]	Thomas [7]	Bulman Bulman [4]
Class	A collection of objects sharing structur & behavior. Implementation of a type.	Description of one or more objects with common attributes & behaviors.	Template specifying object's behavior. Objects are clones.	A type definition. defines both instance variables & methods for objects of a class.	A set of objects with similar behaviors.
Object	An entity expressing precisely define behaviors. Has a state, behavior & identity. An instance of a class.	Object (instance). Encapsulated data attributes & services. An information package & its actions.	A data structure with its encapsulated operations. An instance of a class.	Analog to entity in the world. Encapsulated representation of entities with operations & messages.	Encapsulated knowledge about entities. Implements entities. An entity represents a thing in the world.
Method	Operations declared as part of a class of an object's functions.	Services. Actions or behaviors performed by an object.	Functions or operations defined for a class or object.	Part of an object invoked by a message. Performs the object's actions.	Operations on values of an abstract data type.
Message	Operation done by one object on another.	Message relationship. Descriptive of a processing or service need.	Parameters passed to an object.	Invokes methods. A call.	
Encapsul-ation	Information hiding. Prevents interaction between unlike objects. Hiding of nonessential details from outside world. Modularity.	Containment of a class' information & accessibility. Information hiding.	Operation defined in a specific data structure. Information hiding.	State of an object united with its method	Data are protected from improper access. Forces information hiding.
Inherit-ance	Sharing of structure and/or behavior among classes & objects. Hierarchy of classes & objects.	Shared attributes & services made explicit between or among classes	A directed relationship from a parent to a child. A hierarchy.	Definition of a new object from an existing object with minor differences.	

This paper outlines the background of the object paradigm and its appropriateness to business applications. Then it describes

the object analysis and design of an automated object oriented student housing application. Difficulties with encapsulation are then discussed. Finally, the authors discuss why they chose M for the student housing project.

BACKGROUND OF THE OBJECT PARADIGM

The focus of the object paradigm has been to use a programming language to define application specific functionalities. These functionalities are independently defined, yet are integrated with data structures used to store the instantiations of the objects that depict instances of "the real world" [8]. However, there are few if any non-system (i.e., any application as opposed to a GUI, or system tool) applications available. It was the lack of an available application model that led to the inception of a previous project to detail the pragmatic and conceptual problems with the object paradigm in the modeling and implementation of a student housing program.

As a result, the task at hand was to use the object paradigm as a basis for the analysis and design of a typical business application to establish a model. This would allow the object paradigm to be decomposed from the abstract/conceptual state to one of functional and pragmatic relevance. In other words, we wanted to create a business oriented model that would allow us to document the operating premises and dynamics of the object paradigm.

Historically, the literature has framed the object paradigm within the context of systems issues such as reusability of complex code for graphics and GUIs, the data structures found in the design of databases, and systems tools such as browsers and editors [9]. References to object technology in the most current literature, still embodies this system connotation.

Furthermore, the object technology commercially available (i.e., the so-called *object shells* such as ESIOBJECT by Educational Systems, Inc.), offer both needed and convenient tools for providing GUI utilizing the popular and user friendly *window* and *button* characteristics. In the effort to develop a business oriented application, we found that these commercially available object tools and/or existing applications provided assistance at the conceptual and what we define as the system level.

OBJECT ANALYSIS AND DESIGN OF AN OBJECT ORIENTED STUDENT HOUSING APPLICATION

The student housing application that we chose to model was initiated by a housing staff member in response to a request by a group of faculty , undergraduate , and graduate students for a project of suitable complexity. The application involves the design, development, and implementation of an automated object oriented housing selection/allocation and management system. There are two principal functions which the application performs.

The first function is to allow staff members to define particular constraints for rooms, floors, suites or buildings. Constraints may be applied at any level. Examples include having a building designated as Freshman only or a floor designated for male students only or a particular suite designated for members of a particular fraternity or sorority.

The second function is to allow students to select their own room visually. The student can look at a map of the campus, view a particular building or floor, and eventually designate the room they wish to be assigned. If all constraints applied to the room are satisfied, the room will be assigned.

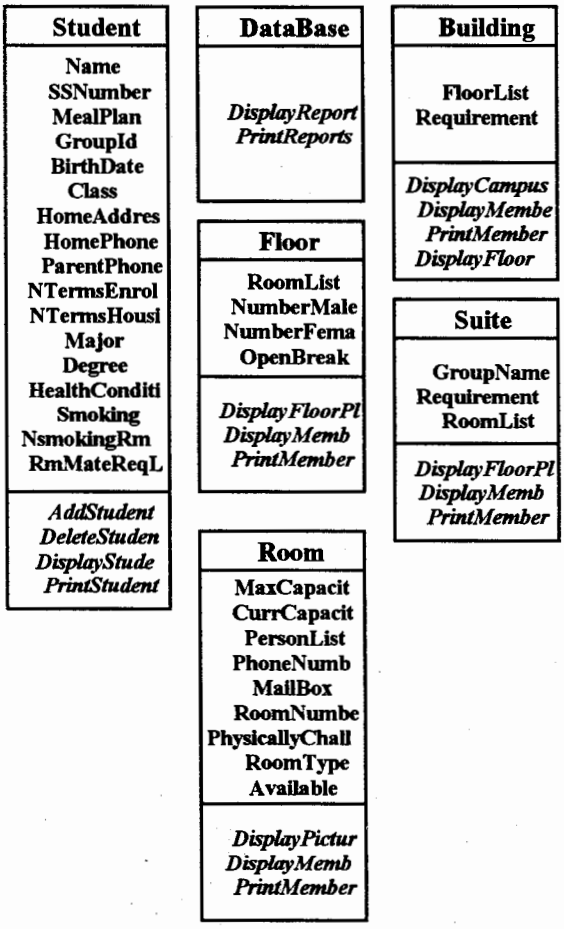
The object oriented features which are enabled are provided principally by two routines. The first, ^method, is designed to call any method in any object. If that method is not present, the ^method routine searches all antecedent classes of the desired object to execute the desired method. Thus "general-specific" inheritance is implemented and polymorphism is facilitated.

The second routine $$^attrib is an extrinsic function to find an attribute of any object. If this attribute is not defined in the target object, $$^attrib searches any other objects to which the object is assigned. This implements "whole-part" inheritance.

Thus, a Room can report attributes of its assigned floor and building to see if a student meets its constraints. Each object stores data in a branch of the M global assigned to it. This data can be imported, exported, or linked to other college information systems. Figure 2 portrays the classes of objects

with attributes and methods that may be applied to the class objects used in the Student Housing system.

Figure 2: Classes with Attributes and Methods of Student Housing System



DIFFICULTIES WITH ENCAPSULATION

We encountered difficulty in the issue of encapsulation. *Encapsulation* is the protection of a program component or data, here *objects,* from improper access and use[15]. This was a critical point in the development of our application within the context of the object perspective. We soon discovered an apparent inconsistency with the *traditional* object paradigm which assumes physical encapsulation of objects. This renders non object oriented systems incompatible with object oriented systems. We found it both unworkable and inconceivable to incorporate (encapsulate) all relevant data within the appropriate objects. For example, it would be ineffective to have each of the nine hundred students' data (i.e., social security number, first, middle and last name) to be combined with the programming code that defines the object. It is also impractical to expect to redo non object systems as they currently exist as object oriented systems.

While the traditional object paradigm regards the code that defines the object and the data utilized by that code as inseparable, we found this unacceptable and unrealistic in a business oriented environment utilizing extensive databases. In order to satisfy the requirements of the student housing system and the object paradigm we solved the encapsulation problem by operationally distinguishing between *physical* and *logical* encapsulation.

Physical encapsulation is the traditional perspective on viewing the ownership of the data by the object. Logical encapsulation embodies the same conceptual inseparability of object and data, yet is consistent with the physically separate databases common in business environments. Although the data is not physically part of the object, it is logically tied to the data contained elsewhere (i.e., a database). It is in fact as inseparable as is in the case of physical encapsulation to the extent that if the object is deleted, the data as it was logically defined is also deleted. However, the advantage is in that the data is still physically present and available for other applications. This is the conceptual equivalent of a *data view.*

It must be stressed that physical encapsulation as an integral part of the traditional object paradigm historically developed with systems applications as a focus. The parallel development of large scale centralized databases and the need for an organization to have access to such data questions whether the traditional object paradigm is compatible with contemporary organizational information processing needs.

M AND THE STUDENT HOUSING PROJECT

The technical considerations of a programming language that supports logical encapsulation and allows for the conceptual integrity of the object paradigm must be considered. That is why we chose M (Formerly MUMPS -- Massachusetts General Hospital Utility Multi-Programming System) as a language that fulfills the technical requirements of having the characteristics of an operating system, a programming language that can be highly structured, and an architecture that incorporates an integrated data structure facilitator (database) to act as our developmental language [11]. In particular, one feature of the language, the global, is uniquely suited to object programming and the issue of encapsulation.

As a result, there are three basic ways of treating object encapsulation in M. The authors feel that first could be called

true physical encapsulation. In this case, a tool is needed which allows object instances to persist as permanent entities or globals. Then both the attributes of the objects and their methods are treated as structures saved within the object. The actual physical representation of the object is not generally known to the programmer, and the object can only be dealt with in the context of the object oriented application. This also requires a run-time module or translation tool to execute the methods.

The second method of encapsulation is pure logical encapsulation. All data for an application would be contained in one *database* object which is represented as a large (rather traditional) M global. Each of the classes in the application system are represented as routines which send messages to the database object which manipulates the global. Thus, this application could easily coexist with other non-object M applications currently using the database. Thus, the database is not physically encapsulated.

The third method of encapsulation is a hybrid of the above two. In this method, each class is represented as both a routine and a global. The global contains all the instance values of all attributes of a class, and the routine contains all methods of the class. This separates the data into separate physical objects (globals) which represent a class, and yet it is still available for external non-object routines to access.

We chose the third method of encapsulation for our student housing model. The class BUILDING will be used to illustrate this method of encapsulation because it is the simplest one of the objects. As mentioned before, an object of class BUILDING along with another class called FLOOR and a third called ROOM which deal with more specific component objects. These classes together wholly describe the state of all dormitory buildings on campus.

The data for BUILDING is encapsulated in the M global ^bldg and is indexed by the building name. Specifically,

^bldg(name,1) -- filename for bitmap of building's picture
^bldg(name,2) -- list of floors belonging to building
^bldg(name,3) -- list of requirements for those in bldg (for instance
CLASS=FRESHMAN,SEX=MALE,GROUP=ATHLETE etc.)

The methods for BUILDING are encapsulated in the routine hsebldg. These methods use DT-Windows for communication with WINDOWS 3.1 windows, dialog boxes and picture displays. Figure 3 lists some of the methods in the routine **hsebldg** that are used to support encapsulation.

Figure 3: hsebldg routine

```
add + getdata  -- adds a new building
display  -- displays a floor map of the building
prmembers  -- prints list of students assigned to building (by
calling each floor in turn to print out its students)

hsebldg;dfw;11:27 PM  21 Feb 1993;
  ;
  ; Methods for Class Building
  QUIT
exists(name) ; returns 1 if name exists, 0 otherwise
  i '$d(^bldg(name)) w !,"Unknown building ",name,! q 0
  e  q 1
add;  adds new building
  w !,"Selected: Adding New Building...",!
  s did=$get(dialog(wid,mid)) i did="" BREAK
  u DTW
  w /wuse(1,1,2),/wsettext("") ;Clear Name
  w /wuse(1,1,4),/wsettext("") ;Clear bitmap file
  w /wuse(1,1,6),/wsettext("") ;Clear floor list
  w /wuse(1,1,8),/wsettext("") ;Clear clear requirement list
  w /wuse(wid,did),/wopendialog
  u 0
  q
getdata; Results of the add dialog box
  w !,"Adding New Building Dialog",!
  u DTW
  k c
  s stop=0
  f  d  i stop q
  . w /wgetmessage(.x) i x="" q
  . i +x=5 s stop=1
  . i +x'=22 s x="" q
  . s data=$p(x,$c(22),5,999)
  . s n=+$p(data,$c(22),2) ; number of messages coming
  . f i=1:1:n f  d  i x'="" q
  .. w /getmessage(.x) i x="" q
  .. i +x'=22 s x="" q     ; only control messages count
  .. s id=+$p(x,$c(22),4)  ; control id
  .. s c(id)=+$p(x,$c(22),6,999)   ; data
  s name=c(2)
  s ^bldg(name,1)=c(4)
  s ^bldg(name,2)=c(6)
  s ^bldg(name,3)=c(8)
  u 0
  q
```

```
display(name) ; show floor map picture of building name
  w !,"Create Child-Window displaying building ",name,!
  if '$$exists(name) q
  u DTW
  w /wuse(wid)
  w /wcreate(201,5,5,400,250,name,2,8+16+64+256)
  s bmp=^bldg(name,1) ; define bitmap file
  w /wuse(201),/wicon(1),/wdrawbitmap(0,bmp,0,0)
  u 0
  q
prmembers(name) ; print list of students in building
  n c,i
  s c=","
  if '$$exists(name) q
  f i=1:1 s floor=$p(^bldg(name,2),c,i) q:floor="" d
  . d prmembers^hsefloor(name,floor) ; call prmembers in
  floor
  q
```

CONCLUSION

What began as a directed study seminar developed into a research project which not only uncovered a greater number of questions than it provided answers but also led to greater application challenges. Important issues that surfaced included: the appropriateness of the traditional object paradigm to non-system related applications, the usefulness and effectiveness of the commercially available object *shells*, and the resolution of conceptual/practical considerations in the design and development of object oriented application systems.

Although the business oriented application in object form is distinct from the physically encapsulated data/object model, the use of the M programming language, with it's inherent global data structure, allows for a hybrid logical encapsulation that provides object advantages within a database driven business environment. As a final point, the time in developing an object oriented business application seems inordinate to its final form. However, it forces the designer and/or application maintenance person to view the application from a holistic perspective. This enhances the ease of maintenance and modification of the application from which the economic benefits of the object oriented system can be realized.

REFERENCES

[1] Gall, Don. (1992). "Object Oriented Programming, MUMPS and the Real World." OOPSLA Proceedings.

[2] Rentsch, T. (1982). "Object-Oriented Programming." SIGPLAN Notices, Vol 17 (12), p.51.

[3] Booch, Grady. (1991). Object Oriented Design with Applications. Redwood City, CA: The Benjamin/Cummings Publishing Co.

[4] Bulman, David W. and Bulman, Erin Kathleen. (1992). "Things, Objects, Entities, and Knowledge." Computer Language. January 1992, pp. 44-48.

[5] Coad, Peter. and Yourdan, Edward. (1990). Object-Oriented Analysis. Englewood Cliffs: Prentice Hall.

[6] Page-Jones, Meilie, Constantine, Larry L. and Weiss, Steven. (1990). "Modeling Object-Oriented Systems: The Uniform Object Notation." Computer Language. October 1990, pp. 69-87.

[7] Thomas, Dave. "What's in an Object?." Byte. March 1989, pp. 231-240.

[8] Kohun, Frederick G. and Skovira, Robert J. (1993). "The Analysis and Design of an Object Oriented Payroll Application: A Validity Test for the Object Paradigm." NEDSI Proceedings.

[9] Schuller, G. (1991). "A Proposal for the Future Development of MUMPS as an Open, Distributed, Extensible, Object-Oriented Data Processing System." Unpublished paper. University of Wurzburg Computing Center.

[10] Aranow, Eric. (1992). "Object Technology Means Object-Oriented Thinking." Software Magazine. March, 1992.

[11] Harvey, William and Kohun, Frederick. (1993) . "MUMPS." Encyclopedia of Microcomputers. New York: Marcel Dekker.

[12] Rumbaugh, James et al. (1991). Object-Oriented Modeling and Design. Englewood Cliffs, NJ. Prentice Hall.

[13] Firesmith, Donald G. (1993). Object-Oriented Requirements Analysis and Logical Design. New York. John Wiley & Sons.

[14] Coleman, Derek et al. (1994). Object-Oriented Development: The Fusion Method. Englewood Cliffs, NJ. Prentice Hall.

[15] Coad, Peter and Yourdon, Edward. (1991). Object-Oriented Design. Englewood Cliffs, NJ. Prentice Hall.