8-16-1996

Society of Objects

Bob M. Czech
BMC Communications, Inc., bmc@eies.njit.edu

Ajaz R. Rana
New Jersey Institute of Technology, University Heights, rana@cis.njit.edu

Society of Objects

Bob M. Czech
BMC Communications, Inc.
P.O. Box 2843
Clifton, New Jersey 07015
bmc@eies.njit.edu

Ajaz R. Rana
New Jersey Institute of Technology
University Heights
Newark, New Jersey 07102
rana@cis.njit.edu

Introduction and Motivation

Generally, organizational processes are carried out in the form of a policy which is then rewritten as a specification for the development of an information system that represents those processes. Our view is that these changes in rules and policies can, in themselves, be developed and distributed as objects which have an immediate effect upon the organization without having to go through the development system. The static code that is written to carry out the organizational behavior is information in and of itself and should be treated as objects. Coupled with the static coding in business applications, is the lack of the ability to share fine grained objects. For instance, a formula used in a spreadsheet cannot be used in another numerical or non-numerical (word processing) application. For this to happen, a change in thinking must be made from manipulation through applications to manipulating information objects directly.

However, much current information systems thinking is in terms of object oriented implementation techniques and how these can be applied to improve the software development process. A chief point in this thinking is the reuse of code. Objects were originally destined to be used in the hands of the user rather than covered under disjoint applications. Much of this concept of objects has been lost in favor of providing improvement to the programming community (Kay 1993). Object orientation has also been used as a modeling technique to describe organizational models that are then translated into typical business information systems.

This paradigm shift is manifest in a model that follows object oriented techniques in an environment targeted at information intensive organizations. This model emphasizes end user objects, accessibility in terms of random retrieval, and organizing capabilities.

Proposed Model

The proposed model combines three ideas:

1. Vannevar Bush's (Bush 1945) original vision of a memex as a repository of information. The owner of the repository provides information that the system ingests and provides the cross linking of this information to create *associative paths*.

2. The organization as a system that provides an output to its customers or clients. The system is "programmed" by the rules, norms, and culture of the organization as defined by the higher levels of management above those directly responsible for the organization's output.

3. A self organizing system based on classification systems as defined by Brachman (Brachman 1990). These systems are able to classify objects based on their properties.

From this we derive an infrastructure to support the organization. This infrastructure consists of four general layers: technical, object, collaborative, and organizational. Beyond that, the needs of particular industries guide the design and development of additional layers. The latter three all contain similar, but ever increasing capabilities. The object level provides the base, but is also representative of the capabilities found in higher levels. For instance, universal addressing is present in the object layer, but is also present in the layers above it. Each layer, while having the same general capabilities molds the capability into the context in which that layer is operating. Due to space limitations, we restrict our discussion to only two layers: technical and object. The higher levels are described in the complete version of this paper.

Technical: Responsible for the lowest level capabilities such as hardware, networking, and basic input/output capabilities.

Object: Takes place of current day operating system providing the kernel operating capabilities and the information organization. It is here that all the storage management functionality is implemented in terms of objects.

An object is the basic aggregate in the model. Objects are composed of attributes and methods. The attributes, themselves, comprise the basic unit of information in the model. An attribute can exist on its own in definitional terms, but can only be instantiated as part of an object. An object can take any number of attributes and is not confined by any class structure. Attributes may be added or removed during the course of an object's lifetime. Objects become members of classes either explicitly and/or by fulfilling class membership rules defined in class objects. This is important for the following reasons:

• Information systems are highly volatile.

• An object can be included in many contexts in different roles. These uses cannot be made a static property of the object as it passes through its lifetime.

• An object, through its lifetime, will take on new capabilities and release old ones.

Each object receives an unique identity which makes it easily accessible throughout the system. This is the most specific way to address that object. Each attribute may be externally named or unnamed. Externally named attributes may be used as a means of addressing to retrieve the data contained in the value of that attribute.

A key attribute of the object environment is the ability to address information (Engelbart 1984). This very key point is lacking in current information systems by not permitting the direct referencing of data. Data is referenced through successive screens or selections made on a document or worksheet that is visually in front of the user. Some primitive addressing occurs in application specific contexts. For instance, a worksheet may allow a user to address a cell in that worksheet and name a particular area. Similarly, a word processor may allow a user to create bookmarks to a particular page. However, there is no universality of the address capabilities. In the rush to visual environments the capability to have effects on information that is not visually in front of the user has been overlooked.

Addressing is only relevant within a structure containing a list of objects. Objects in a structure must have both absolute and relative addresses. In addition, an object has a constant address by which it is known within a structure. At the highest level, the object environment itself, this is done in terms of a serial number that is associated with that object throughout its lifetime. Each structure may have its own addressing scheme and these may be combined recursively to create complex structures.

Attractor Objects

A significant aspect of the model is the Attractor Object. These objects have the characteristic that they can attract other objects into themselves. Attractors are similar in nature to the definitions provided in KL-ONE class systems such as Classic (Brachman 1990). These objects define a class in terms of what an object's attributes must satisfy in order for that object to be a member of that particular class. In this model, we extended this to mean that an object "attracts" an object to it if the object satisfies the rules.

An attractor defines a set of rules for inclusion. When an object is created or modified, it is tested against these rules. If the rules are successful, then the object is sent to the attractor object which then ingests this object in a manner defined by that attractor object. The attractor permits systems to be extremely dynamic. This is important to allowing end users to work directly with objects. The ability of attractors to take in objects allows objects to be created independently and autonomously.

When speaking in terms of these layers, it is not meant that layer *n* is built using the classes created from layer *n - 1*. Rather, layer *n* is built using the objects that are instantiated in layer *n - 1*. While these layers could be implemented with the same language, it may be desired that each layer provide additional language abstractions to the higher levels.

Applications, in this context, are just aggregations of objects with a view over that aggregation. Capabilities are contained in the objects themselves. The application adds no new capabilities.

Comparison to Existing Object Environments

There are a number of object environments that currently exist to solve various problems associated with distributed object management. The three main environments are CORBA by OMG, OpenDoc by Cilabs, and OLE by Microsoft. CORBA is aimed at the cross platform, heterogeneous object environment. OpenDoc and OLE are aimed at productivity tools utilizing embedded objects in documents. We will compare how these are different from the model proposed here and how the model rectifies the shortcomings that we highlight.

CORBA is an object communication model designed to permit heterogeneously implemented objects to be accessed.:

CORBA is an object interface mechanism as opposed to an object content infrastructure.

Objects communicate, but do not migrate

Objects are not homogeneous network wide with respect to basic object capabilities.

OLE defines a model whereby an object of differing format can be embedded in another object. The embedded object is operated upon by means of an interface that the application to which the embedded object belongs provides. OLE falls short of true object expectations by tying an object to a specific application rather than being independent of any application or at least to any class of application. For instance, you cannot open a spreadsheet using any spreadsheet application, nor access cells as independent entities from the spreadsheet itself. This lack of a consistent underlying object structure differs greatly from the model that is proposed by still centering on applications rather than the data. OpenDoc still operates under the paradigm of applications, but makes a finer grain delineation in attempting to provide a solution to the problem of the mega applications that are now written.

Implications and Conclusion

A number of changes occurs in the view of information systems by using the model presented above:

By maintaining all organizational behavior distributed amongst objects, there would be no need for applications, which today, are used to apply the business logic to the data contained in traditional database structures.

The infrastructure is the database that lives in the database and concentrates on the manipulation of data instead of applications.

Processes are defined by their dependency on other objects. Processes are not defined by a serial series of steps which must be completed in order, but rather requirements for a task activity to be completed regardless of order.

Objects that are aggregated together would provide a composite behavior, the logic used to glue them together in terms of the user interface is the only additional code.

Each individual organizational unit would be responsible for creating their own objects to add to the object space. The need for large, encompassing information systems would be eliminated. Each unit could also develop their own language to manipulate the objects in their particular domain.

Each unit would then be able to debug their organizational structure as they implemented their objects. This differs from the current situation in which organizational problems that are found in system design specs are probably not relayed back to the originating organization.

Information development and maintenance becomes a universal responsibility.

Removing file structures would eliminate the problem of understanding among naive users.

Having a highly mutable object structure, changes in field composition do not effect other objects. Today, changes in object structures cause whole applications to be recompiled.

Information has more dimensionality. In other words, information content is not restricted to that which is visual on the printed page, but also to all the objects which comprise the knowledge behind it. So, the printed format is just a particular view onto the complex of objects which make up the true structure behind it.

By providing objects to the end user, they may readily combine these objects to complete tasks that under application oriented systems would not be allowed, or would have to cause a software maintenance cycle to be incurred.

The proposed model here attempts to create a better way to create information systems rather than create a "better old thing". The model not only increases the rate at which an organization can react to change and its ability to ingest intelligence from its environment, but provides support to many of the ideals of current business management: accessibility and empowerment.

References

Bush, Vannevar (1945). *As We May Think*. Atlantic Monthly, Vol. 176 No. 1, 1945.
Brachman, Ronald J., McGuiness, D., Patel-Schneider, P., Alpern-Resnick, L., Borgida, A. (1990). *Living with CLASSIC: When and How to Use a KL-ONE Like Language, Principles of Semantic Networks*. Morgan Kaufman Publishers, 1990.
Engelbart, Douglas C., (1984). *Authorship Provisions in Augment.*. Proceedings of the 1984 COMPCON Conference, San Francisco, CA., February 27 - March 1, 1984.
Kay, Alan C. (1993). *The Early History of Smalltalk*. History of Programming Languages II, April 1993. Sigplan Notices 28(3) March 1993.