

## An Object-Oriented Strategy for Banking

J DARATOS<sup>‡</sup>, AL STEENKAMP<sup>†</sup>

<sup>‡</sup> *Systems Building Services, ABSA Bank Ltd*

*PO Box 781700, Sandton, 2146*

*Facsimile: 011-884-1586*

<sup>†</sup> *Department of Computer Science and Information Systems*

*PO Box 392, Pretoria, 0001*

*Facsimile: 012-429-3434*

**Abstract:** Many information strategists today agree that the *Object-Oriented Technology (OOT)* has particular requirements that should not be omitted from a traditional *Information Technology (IT)* strategy. An *Object-Oriented Strategy (OOS)* model, developed using an *Object-Oriented (O-O)* methodology and addressing the particular requirements of OOT, can provide invaluable guidelines as to the deployment of OOT in any organization. For organizations that are traditionally conservative to technological change, an OOS can also provide guidelines for dealing with the cultural and organizational changes that inevitably accompany the transition to OOT. This is especially so in the banking domain where the need to protect the wealth entrusted by its customers (this need can be traced back to Babylonian times) is a contributor of the conservative approach to change. This paper addresses the importance of an OOS for banking through a brief history of banking and information strategies, software complexity issues that influence information strategies, current O-O methodologies and standards, and a brief discussion showing how an OOS can be derived using an O-O methodology.

**Keywords:** *Bank, Banking, Information Technology, Object-Orientation, Object-Oriented Methodology, Object-Oriented Technology, Object-Oriented Strategy.*

**Computing Review Category:** K.6

### 1. Introduction

An establishment receiving money for the purpose of being lent out on interest, or returned by exchange, or disposed of to profit, or to be drawn out again as the owner requires it, is referred to as a *bank*. The term *banking* is used to describe the receipt of money from a bank's customers and in making payments on their behalf. In addition, banks also undertake a wide range of activities connected with domestic and international finance. Many banks today are also competing against one another by providing various kinds of services to customers backed by sophisticated computer technology (e.g., automatic cash dispensing machines, online credit card authorizations, home banking, etc.).

Early banking functions of an elementary nature can be traced back to remote times. In ancient Babylon, for example, private persons entrusted their money to the care of the temple 'treasury'. In ancient Rome the *argentarii*, or 'silver dealers', changed money of different types and origins. Mention is also made in the *Bible* of usurers and money-lenders. Even with these early forms of banking activities, however, *information* played an important role in the control of banking matters. As primitive as banking functions might have been, *strategies* (i.e., plans of action) for implementing the best available method for information storage and retrieval were formulated and applied accordingly.

## 2. Evolution of Information Strategies

As far back as 3500 B.C. Babylonian merchants imprinted records of wealth on clay tablets known as *cuneiforms*. A person strategizing for information storage and retrieval, therefore, had a simple choice to make; i.e., use either cuneiforms or mental storage and retrieval. This kind of strategy, however, was not to remain so simple. Through the use of language (invented by the Phoenicians at about 1500 B.C.) and mathematics (that can be traced to about 3500 B.C.) people have continuously refined and expanded their knowledge and understanding of themselves and their environment. Indeed, language and mathematics became the basis for all subsequent developments in data and information processing that, in turn, led to a demand for more sophisticated information strategies.

While the evolution of data and information processing has been characterized by advancements in both theory and technology, each successive technology innovation has been accompanied by increased complexity in understanding its essence and usage. It is much easier, for example, to train a person to use an *abacus* (perhaps the simplest calculating device in use since 3000 B.C.) than a *digital computer* (a modern, yet very complex, calculating device) and its related technology. *Computer software*, in particular, is perhaps more complex than any other construct made by human beings. Similarly, the formulation of a strategy for developing software that produce and maintain quality information implies a complex task.

## 3. Software Complexity Issues

The consequences of software complexity in building software products is an important issue to be addressed by the information strategist. Without appropriate measures that reduce software complexity the chances for producing low-quality products, deadlines not being met, budget overruns, and high maintenance costs are increased. In fact, such problems were recognized and collectively termed the *software crisis* as far back as 1968 in a NATO software engineering conference.

Although any organization would be advised to take heed of the software crisis (that still prevails today) and strategize accordingly, banks should be even more concerned in this regard. Accounts are no longer stored on cuneiforms and the information required by customers is no longer restricted to the amount of wealth they possess. Today, customers are provided with a diversity of banking services supported by a sophisticated computer technology that includes complex software. Unless this technology is utilized effectively to deliver quality information on time to the customer, there is no guarantee that the customer will not move on to the next bank offering a better service. Software complexity, therefore, should be addressed by the information strategist not only to provide services to the customer, but also to maintain an adequate customer base.

According to Brooks [2], high-level programming languages, time sharing, and software development environments (such as UNIX) have been *silver bullets* (technology advancements) that have helped but have not overcome the problems associated with the complexity of computer software. Banks have been, and still are, using such silver bullets to deal with software complexity. Brooks [2] also predicts that other silver bullets, such as correctness proofs, object-oriented design, Ada, and artificial intelligence, will alleviate some (but not all) of the complexity problems.

Many banks today are particularly conservative when it comes to implementing the latter silver bullets. A new, strange way of developing software is seen as a potential disruption (and perhaps loss) of a customer's nest-egg and should not be adopted without proper evaluation and control. In effect, the ancient principle of protecting customer wealth still applies today. This also means that unless banks find appropriate ways to deal with software complexity, they run the risk of lagging behind in a fiercely competitive technological world.

## 4. Dealing with Software Complexity

Since complexity is an inherent characteristic of software, it is not possible, at least for now, to eliminate the software complexity problem. What is possible, however, is to formulate an information strategy that addresses this problem directly. For example, such a strategy could target an *Information Technology (IT)* architecture that embraces a new software development paradigm conducive to reducing software

complexity as well as software development costs.

Perhaps the first time that banks had to deal with software complexity of a sizeable magnitude was in the 1960s, the so-called *real-time era*, when the feedback characteristic of a system became predominant by requiring that system results be available immediately. A *real-time system* satisfying this requirement, however, exhibited patterns of complexity not explainable by the size of the hardware alone [5]. A program required to poll and respond to outlying terminals, for example, had much more involved logic than a program written to control only the card reader. One strategy suggested for this *multiprogramming* problem was to introduce a control program as a work scheduler and interrupt processor that would service a number of operational programs dealing with customer accounts (e.g., inquiring and updating savings accounts, calculating interest, etc.).

With the introduction of network control programs (such as IBM's Virtual Telecommunications Access Method) and databases (such as IBM's Information Management System), the complexity of networking and information storage and retrieval was reduced to a certain degree but, at the same time, made the task of formulating an appropriate information strategy more difficult for the information strategist; e.g., the increasing number of software products offered by vendors, the variety of systems development methods and techniques that began to emerge in the 1970s, the continued improvement in hardware, and the sophistication of computer networks increasingly complicated decision-making as to the IT direction that an organization should follow.

## 5. Current IT Strategy Trends

Many organizations today are faced (more than ever) with IT issues that involve rapid change and adaptation to new ways of software development. Banks, in particular, have traditionally opted for safer, more established technologies that do not present a threat to customer accounts; thus, many banks have been, and still are, investing in technologies provided by the bigger and financially sound suppliers such as IBM and NCR. It is not surprising, therefore, that hardware such as the newer IBM 390 range mainframes and PS/2 personal computers, operating systems such as IBM's MVS/XA on mainframes and OS/2 on personal computers, networking solutions such as IBM's LU6.2 protocol and LAN/DP, databases such as IMS/DB and DB2, and transaction processing monitors such as CICS and IMS/DC are still integral components of many a bank's IT strategy.

Structured development methods and techniques, hierarchical and relational database design, and an organizational structure separating the business from the data processing function have also dominated the banking environment over many years (roughly since the real-time era). In fact, this kind of development approach has ingrained, over the years, a culture that is resistant to change. Any new development method or technique, although scrutinized with interest, is usually rejected or implemented in a 'safe' environment (such as a background office function). The *Object-Oriented (O-O)* approach to systems development, however, has aroused great interest both in the banking and other environments.

## 6. Strategizing for Object-Oriented

The current surge of interest in the O-O approach to systems development is probably due to claims that it is a silver bullet with superior benefits, especially when it comes to software reuse. In formulating a strategy for the implementation of an O-O development environment, however, the strategist should note that there are at least fifteen O-O methodologies for systems development, the most popular from Rumbaugh et al [6], and then Wirfs-Brock, Shlaer-Mellor, Booch, Coad and Yourdon, and Martin-Odell. Metamodels have also been developed, and are still being developed, to support the automation of O-O methodologies.

The strategist should also note that O-O standards have already been proposed by the *Portable Operating System Interface (POSIX)* reference model, the *Open Systems Interconnection (OSI)* reference model, the O-O extensions to SQL, known as *SQL3*, and Microsoft's *Object Linking and Embedding (OLE)* mechanism [7]. The *Object Management Group (OMG)*, however, appears to be prominent in this area with standards proposed by its *Common Object Request Broker Architecture (CORBA)*. The OMG was established in April 1989 in an attempt to overcome the barriers between the *Object-Oriented Technology (OOT)* and users. The OMG is a non-profit, member-sponsored organization dedicated to establishing open

standards in the area of OOT. OMG realizes that the OOT requires a different view of analyzing problems and provides a powerful and efficient means of enabling solutions. According to OMG, the OOT can be valuable in providing easy information access to all levels of people and is demonstrated by the fact that it is currently reducing application development time and maintenance at many user sites.

Although the OMG can be an invaluable source of strategic information, it would be useful to have a strategy model as a reference that aids organizations in their efforts of migrating towards an O-O development environment. Such a model would also be very useful for a banking environment, providing those strategic solutions that would minimize customer service disruption and maximize customer account protection. The next section describes how such a model can be derived using O-O methods and techniques without claiming that other IT strategies do not exist or are not perhaps more suitable in strategizing for OOT.

## 7. The Object-Oriented Strategy Model

The derivation of the *Object-Oriented Strategy (OOS)* model can be accomplished by applying an O-O methodology. Daratos [3], for example, has provided a detailed explanation of how the conceptualization, analysis, and design steps of the Booch [1] methodology can be used to construct such a model. A brief overview of the steps needed to derive the OOS, as proposed by Daratos [3], is given in the following sections.

### 7.1 Conceptualization

The main requirement for an OOS model is that it should describe a system of appropriate *responsibilities* and *activities* needed to implement OOT in a particular organization (a bank is also an organization). Other important requirements may be listed as follows:

1. Descriptions of functional responsibilities of players in the key areas of the OOS must be included in the model.
2. Functional dependencies of players and key areas in the OOS must be modelled as well.
3. The responsibilities and activities must be based on sound academic and industrial knowledge.
4. It must be possible to apply the model in any organization.

Note that an additional, but not necessary, requirement is that the OOS model should be conducive to automation; i.e., it should be possible to build a software system (perhaps an expert system) that guides the user through each step in the implementation of OOT.

### 7.2 Analysis

The fourth requirement in the conceptualization step actually provides the basis for proceeding with the analysis step. Firstly, suppose that a *Generic Object-Oriented Strategy (GOOS)* model exists with appropriate strategizing rules for formulating an OOS in any organization (as per requirement #4). Secondly, suppose that a *Complete Strategy Set (CSS)* exists containing appropriate responsibilities and activities needed for implementing OOT in a particular organization (in accordance with the main requirement). Thirdly, suppose that a *Minimal Strategy Set (MSS)* can be created by selecting those elements of the CSS that an organization deems appropriate; e.g., a smaller organization might select a subset of the CSS due to organizational and/or cost-related constraints. Finally, suppose that an *Alternative Strategy Set (ASS)* exists that contains elements not included in the CSS and that it is possible to model the GOOS as shown in Figure 1.

Now, according to the GOOS model, the first rule is that the strategist should acquire a good understanding of OOT and the OOS system before continuing with the next step of strategy formulation. This implies that the responsibilities and activities in the CSS must be reliable, else they are of little or no use to the strategist. *Domain analysis* is thus carried out first, using documented academic and industrial knowledge (as per requirement #3), to identify potential classes and objects from appropriate abstractions. *Scenario planning* is then carried out to describe system functionality using story-boarding techniques called *scenarios* that are captured by a *use-case analysis diagram*. As shown in Figure 2, for

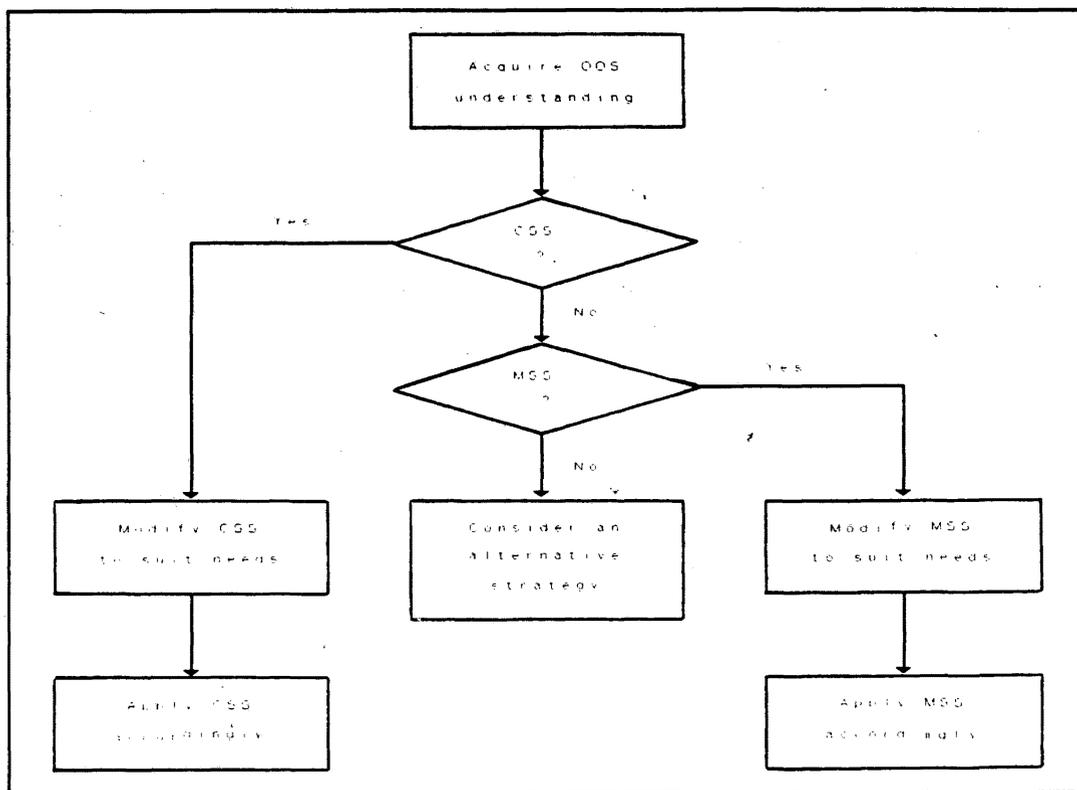


Figure 1. The Generic Object-Oriented Strategy (GOOS) model.

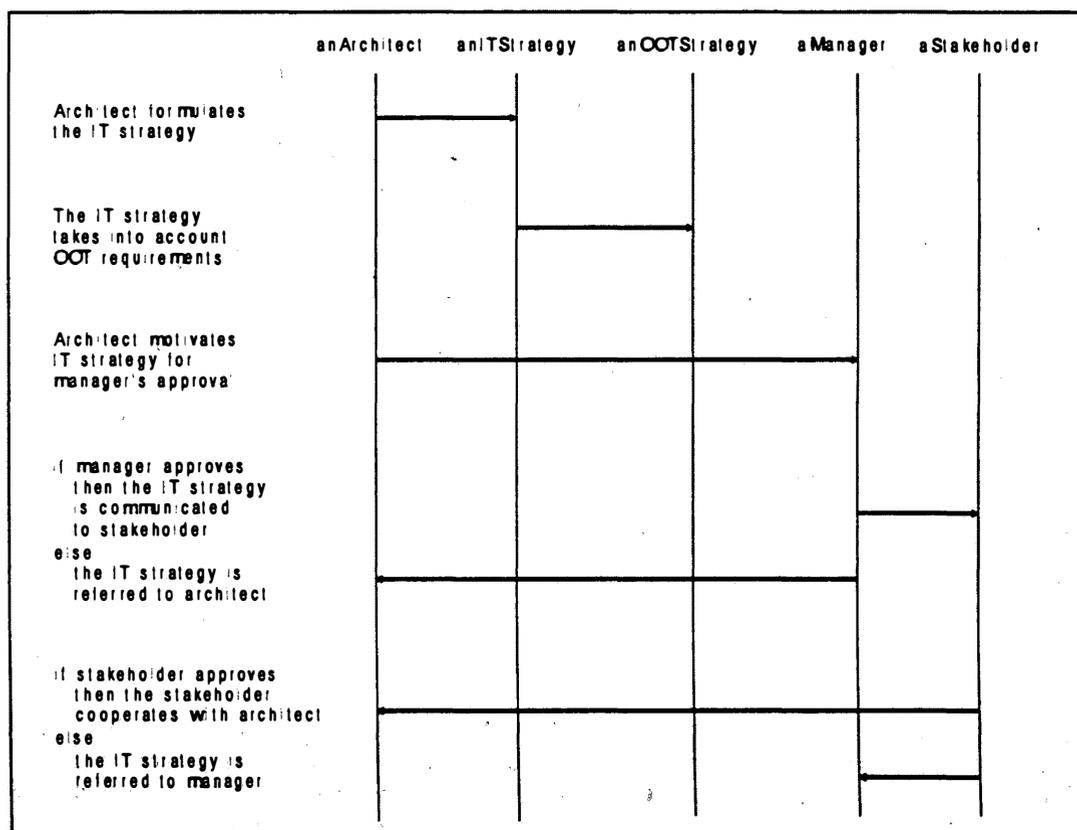


Figure 2. OOS use-case analysis.

example, *aManager* object interacts with *aStakeholder* object, provided *aManager* object approves *anITStrategy* object (use-case analysis diagrams thus comply with requirement #2).

Following domain analysis and scenario planning, it is then possible to produce a *class diagram* showing the key abstractions (players) and their functional responsibilities (as per requirement #1). As shown in Figure 3, for example, an object from the *Architect* class has to cooperate with an object form the *Stakeholder* class. Note that the M to N relationship might apply in the larger organizations, where many architects might be involved with many stakeholders in implementing the OOS (some organizations, however, might prefer a 1 to 1 or even a 1 to N relationship).

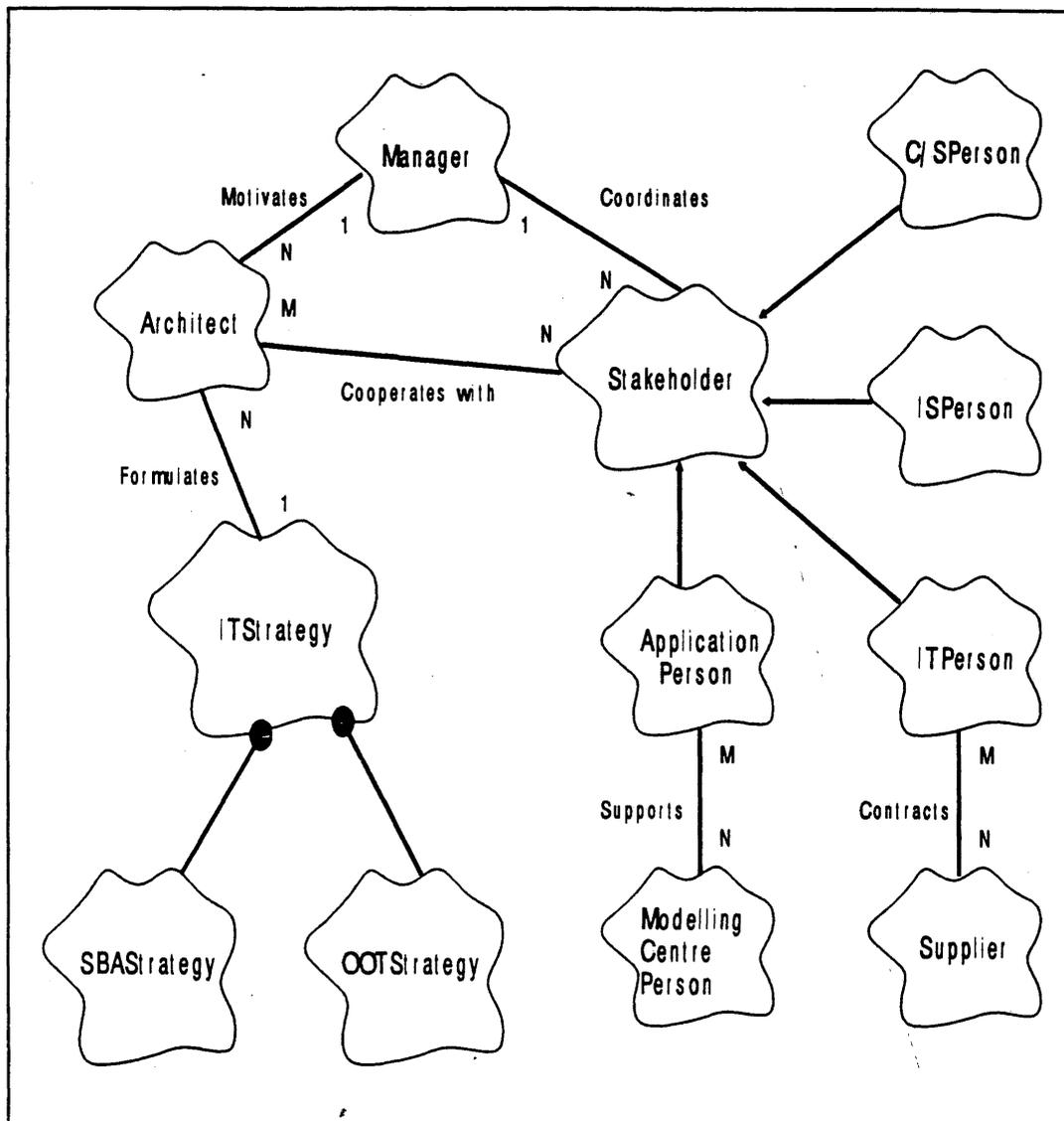


Figure 3. OOS class diagram.

### 7.3 Design

The main objective of the design step is to populate the CSS with appropriate activities that have to be performed in order to satisfy certain responsibilities. In O-O methodology terms, high-level operations (activities) that operate on appropriate attributes (responsibilities) must be defined and assigned to appropriate classes so that an appropriate CSS can be derived. Table 1, for example, shows a subset (or MSS) of a particular OOS for the *Architect* class (the CSS is given in [2]). This particular OOS proposes that the architectural function in an organization should perform three activities (indicated by the operations section of the class) in order to meet three corresponding responsibilities (indicated by the

attributes section of the class); i.e., to evaluate the OOT, to evaluate the reuse technology, and to formulate an OOT strategy. An important point to note here, however, is that each operation defined in the class provides the process (i.e., sequence of steps) through which a particular strategy step can be accomplished. For example, the *evaluate\_OOT* operation might propose a process showing how vendor products are identified, evaluated, and selected accordingly.

<pre> class Architect   attributes     object-oriented_technology     reuse_technology     strategy   operations     evaluate_OOT     evaluate_reuse_technology     formulate_OOT_strategy </pre>
---

Table 1. Architect's activities and responsibilities.

## 8. Conclusions

An IT strategy formulated using an O-O methodology can be an invaluable aid in implementing OOT in an organization. When applied in a banking environment, for example, it can provide both the mechanistic steps needed to successfully implement OOT as well as appropriate guidelines concerning cultural and organizational change. Since an O-O methodology can be used to design an O-O strategy [3] as well as to design a software system, an O-O strategy developed in this way (such as the OOS) must be conducive to automation. The rules provided by a generic O-O strategy (such as the GOOS) can also aid in the automation process, by providing a high-level specification of the actions and options that the strategist needs to select from the CSS.

Given that the CSS contains reliable elements (i.e., responsibilities and activities), the information strategist in an organization can select the entire CSS or a MSS; however, there may be situations in certain organizations that the CSS does not cater for; e.g., there may not be an operation catering for OOT deployment on *Automatic Teller Machines (ATMs)* in the banking environment. In such cases, the GOOS provides an exit point, where the strategist would populate the ASS with new elements that might possibly solve the outstanding strategy problem. The elements in the ASS would then have to be validated for their strategic effectiveness before transferring them to the CSS.

One is tempted to conclude that implementing OOT using an appropriate O-O strategy can lead to very substantial practical benefits in the development of application systems. Despite its glowing possibilities, however, the OOT is still largely unproven in the eyes of users, who have grown to disdain the hyperbole that typically attends new technology. This attitude is particularly true in the banking environment that is traditionally conservative. The expanding product base and efforts aimed at establishing O-O standards should, however, help break down the barriers between these users and what may become the best silver bullet invented yet.

Finally, it should be mentioned that OOT is still immature in many respects and has significant gaps, making banks even more cautious as to its adoption. The most important gaps are the absence of industrial-strength *Object-Oriented Database Management Systems (OODBMSs)* and of development systems for mainframes and minicomputers. These gaps, however, are expected to be filled increasingly over time. By the year 2000 an increasing amount of this functionality will be delivered by O-O software [4]. Information strategists, especially in banking environment, should take heed of this tendency and

strategize for a potential implementation of OOT. In the meantime, the deployment of development on PCs and workstations can offer major advantages over the older development technologies with little or no disruption on business and/or loss of customer accounts.

## References

1. Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Inc., 2nd Edition.
2. Brooks, F.P. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*. 20(4):10-19.
3. Daratos, J. 1994. *Strategizing for Object-Oriented in a Banking Environment*. MSc Dissertation, University of South Africa.
4. Flint, D., Bowler, R., Bush, N., Pawson, R., Quintrell, J., Schmidt, L., Forge, N., & Cooper, J. 1992. *Introducing Object Orientation*. Foundation Research Report 88, CSC Index.
5. Head, R.V. 1964. *Real-Time Business Systems*. Holt, Rinehart and Winston, Inc.
6. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorenzen, W. 1991. *Object-Oriented Modelling and Design*. Prentice-Hall, Inc.
7. The OpenDoc Design Team. 1993. Shaping Tomorrow's Software. *IBM OS/2 Developers Connection*. 5(2).