Modelling Messages as Objects

Hossein Saiedian
A. Ravichandran
Dept. of Computer Science
University of Nebraska at Omaha
Omaha, NE 68182

Abstract

In traditional computer-based message systems, messages are typically viewed as passive entities with no processing power. The users are thus responsible for explicitly creating, routing and processing each message instance. Messages, though, may be viewed as objects which can do processing on their own. With increasing interest in object-based systems, it is not difficult to envisage a system in which messages are viewed as objects with processing capabilities comparable to the objects in object-based systems. This paper explores the concept of "messages as objects" and speculates on the significance and advantages of viewing messages as objects. Research issues in formal definition of an object-based message management system are discussed at the end.

1 Introduction

Computer-Based Message Systems (CBMS), also known as electronic message\(^1\) or mail systems, allow two or more individuals (or entities) to communicate electronically. In these systems, computers serve both to mediate the actual transmission of message between entities involved as well as providing the users with facilities to create, read, route, or destroy messages. The advantages of CBMS can be summarized as follow [2]:

- Allow asynchronous communication in which the expected recipient(s) need not to be ready at the time of message transmission.
- Users are released from geographic restriction.
- Sender's and receiver's times are optimally used.
- Reliable and speedy transport of messages is provided.

Computer-based message systems have been in existence for over twenty years. They were one of the earliest results of research sponsored by the Advanced Research Project Agency [16]. One of the earliest successful CBMS was called MSG [22]. SRI International has published a bibliography, summarizing resource use in computer message systems [6].

The computer-based message systems have proven to be an invaluable communication tool to computer users in various organizations. They have created new forums for exchanging information between individuals who have never met. Furthermore, message systems have become an important area of research in office automation. Since offices are communication intensive environments, computer-based message systems are integral part of any office information system.\(^2\)

2 The Problem

Traditional computer-based message systems are passive systems in which users have to initiate all actions. In particular, messages are passive entities which consist strictly of data with no processing power. In a typical CBMS, users are required to know all the expected recipients of a message and then have to explicitly specify the message routing path and other message management functions. The message route would be point-to-point, i.e., the sender of the message specifies the most intermediate destination from which the next user must specify the next action, if any, and so forth.\(^3\)

\(^1\)We use the terms message and mail interchangeably in this paper.

\(^2\)A discussion of exploration of communication techniques as it is relevant to office information systems is given in [5].

\(^3\)If a distributed list of recipients is specified by the sender, a copy of the message is sent to every member of the list. However,
Even though electronic message systems are more convenient and offer greater flexibility and power than regular mail systems, we believe these systems should comprise more than just elementary editing and transport functions and that their real capabilities should be fully exploited. What we are proposing is that, unlike the traditional mail systems in which messages are treated as passive entities of data with no processing power, a facility should be incorporated into message systems so that messages can be envisaged as active objects with programmable intelligence. In other words, we suggest a framework for message systems in which messages are active and intelligent which, among other things, can

- collect responses from their recipient,
- make certain routing decisions predicated in the programmable intelligence,
- modify themselves at intermediate routes before continuing their route,
- have a "mission" but can learn from their environment,
- return to originator if necessary,
- augment the message sender's knowledge about the recipient,
- actively seek information when necessary,
- autonomously check for exceptional conditions,
- provide additional security,
- etc.

Thus, we philosophize and argue that active message objects are a natural mechanism for building an advanced message management system. Our objective therefore is to capture a notion of intelligent message object which is an encapsulation of an asynchronous loci of activity with high degree of autonomous responsibility and control.

The basic ideas of our proposed model are in some ways very simple, and in some ways very complex, but they are appealing and have high potentials. Many of the features of our proposed model may not be very difficult to prototype but perhaps will be hard to implement. In the following sections we provide some background on the above topics, relate our research to existing work in the literature, elaborate on potential advantages and significance of viewing messages as active objects and discuss the theoretical issues that need to be formally addressed. In particular, we speculate on the complexities of developing a formal framework which is powerful enough to capture semantic properties of local and global behavior of active messages divorced from any particular implementation.

3 Background

Our study of "message objects" is related to the research concepts in the area of object-orientation [15]. The concepts of objects and active messages are not new. These concepts are briefly reviewed below.

Object-Orientation. Object-orientation is a trend towards software development that has emerged in recent years. In this framework, a complex system is viewed as a collection of objects. Each object is considered to be an autonomous and self-contained entity that represents a physical or abstract entity. One main goal of object-oriented framework is to maintain a direct correspondence between the real-world entities and their representations in computer systems being developed so that objects do not lose their integrity and identity. The object-oriented approach provides advances towards software engineering concepts such as abstraction, modularity, reusability, rapid prototyping, and so forth. A recent issue of the Communications of ACM (September 1990) provides several excellent technical articles on object-orientation. Other good references include two books, one edited by Kim and Lochovsky [11] and one edited by Shriver and Wegner [18].

Active Objects. The earliest theoretical work to characterize the behavior of active objects was the actor model [9]. In this model, "actors" and "events" are the fundamental concepts. Actors are referred to computational agents while events mark the arrival of messages at an actor. Actors interact with each other through one actor sending a messenger (or a message) to another actor. The key point in the actor model that Hewitt and Baker presented was that messengers were themselves actors or active objects. Hewitt and Baker's goal was to present a model in which everything was an object and thus provided an untyped theory which would be a generalization of the λ-calculus of Church [4]. However, this approach, taken to an extreme, would lead to an infinite regression, since the only way two actors would interact would be by sending messengers [12]. But if a message itself is
an actor, then one must send it a message, and so on. In 1986 Agha [1] provided a formal definition of the actor model in which messages were no longer treated as actors, but as abstract communications. Agha argued that if messages were viewed as actors, the formal definition of semantics of the actor model would become too complicated.

Active Messages. The earliest work in this area belongs to Vittal [23]. Vittal described a system in which messages are capable of performing certain actions on their own. In particular, messages can tailor their interactions with a user, depending on the responses they receive from the user. Conceptually, Vittal considers an active message to be a single self-modifying entity and allows simple routing specification as part of an ad hoc serial distribution list. This capability is limited, allowing a simple routing specification and no decision criteria. Furthermore, Vittal does not discuss issues related to dynamic routing and problems associated with multiple copies.

Some recent models provide automatic routing facility [19] [21]. These models allow implicit destination specification and thus free the users from explicitly specifying each destination for each message instance sent. Other models, e.g., [3], use a database alerting technique for office activities management and routing destination specifications are formed as actions in alerts. The logical routing model of Mazer [13] does not view messages as objects with capabilities mentioned earlier, but allows a "logical routing" in which the system assumes responsibility for evaluating the current message instance state to yield the next destination, thus freeing the user from the need to direct each instance of a message. In the Imaill system of Hogg [10] a message is allowed to interact with its recipient, and on the basis of the responses that it collects, decides whether to route to further recipients or terminate. In ABSL model of Saiedian and Unger [17], the entire path a message has to travel may be specified by the originator of the message. The intermediate recipients of the message, however, are neither required nor able to (re-) specify the message route. In the Amessae model of McBride et al [14], a static rule-based framework is incorporated into messages that permit a message to determine the routing which it has to undergo. The time-dependent constraints in this model are specified and verified via a variation of temporal logic language.

Most of these models have typically taken a narrow view of an intelligent message management system in which, message routing is typically single hop, i.e., from the sender to the most immediate destination and each user must explicitly interact with each message as it flows around the system or else the routing decisions are static. Furthermore, none of the above models have fully investigated the concept of messages as objects in which messages would have similar capability and power of objects in object-based systems.

4 Significance

A message object, the way we intend to view it, is essentially an abstract data type [8] with instructions to route a message, process a message, and user interactions embedded in the message object's script (i.e., the executable part of message). As a result, a message object can take a role analogous to a processor executing under instructions of a program and thus it can directly execute the actions which are needed to perform the task assigned to it by its originator. For example, when a message object arrives at a given station, it can execute the code associated with that recipient station. The actions may also have time-dependent constraints, for example, if there is specific time difference between the creation time of the message and the current time, the message object would terminate its mission and archive itself in the originator's station. Also, since a message object may have routing information in itself, it can deliver itself to a next station if the current station doesn't interact with it after some period of time. This approach would also provide a better user interface since the users are released from burden of details for routing procedures.

Since messages are objects, they will have an "identity." Thus one can communicate with them. This implies two added features. First, if a user has sent a message, but does not wish the message to be seen by the intended recipient, he can send another message to the first message, asking it to destroy itself, or to come back (provided that the receiver has not received the message yet). Second, two message objects may communicate with each other. This feature allows cooperative work among messages that are part of the same activity.

To provide a more flexible user interface, a message object may have a number of fields similar to the ones proposed by Gehani [7] and Tsichritzis et al [21]. For example, personalized fields which are automatically filled from the sender's profiles and system variables, required fields which must be entered by the sender when creating a message object, unchangeable fields...
which consist of optional data but cannot be changed once entered, *unrestricted fields* which include optional data and can be modified at any time and by any user, *virtual fields* which are automatically computed by the system according to some computations rules or preconditions specified in the message object's script, *ordered fields* can only be filled by the users after some other fields have been filled, *key field* which is for system identification purposes and is generated by the system when message object is created and may not be modified by the users, *lock fields*, which if filled, result in certain other fields being protected from modification, *invisible fields* which are invisible to users who do not have access rights to read/modify a message body, etc. The need for some or all of the above fields can easily be justified. For example, the *invisible fields* essentially enforce the *access rights*, i.e., they specify users authorized to access, create, destroy, copy, and/or modify a certain message objects. The access rights might be at field level (e.g., certain users may not modify a specific field on a message object) or at the entire message class level (e.g., certain users may not create certain message objects).

5 Computational Model

The message system consists of one or more sites. Each site has computational facilities and in addition facilities to communicate with the other sites. One of the computational facilities at each site is the mail manager. The mail manager is the interface between the users at the same site or at other sites. The mail manager provides the user at a site facilities to send and receive mail. Each mail message is a 2-tuple consisting of not only the data sent by the sender but also the mechanisms (or operations) to interpret the data. In other words, each mail can be viewed as an instance of an abstract data type containing data and operators for this data.

To send a mail, the user sends the mail data and mail operators appropriately encapsulated to the mail manager along with the name of the intended recipient(s). The mail manager uses the communication facilities to transport this mail if required. At the receiving site, the mail manager receives a mail message from its communication facility. The mail manager then informs the recipient of the arrival of a mail message. When the recipient desires to read his mail his request to the mail manager is passed to the received mail by the mail manager. The received mail then interprets the recipient request as an operator and processes this request using the mechanism supplied by the sender on the data supplied by the sender. The results of the processing are then sent to the recipient (either directly or through the mail manager). Since reading of the mail initiates a computation (coded as part of the mail), further computations may be generated with or without the knowledge of the recipient. For example, the sender may code as part of the read command an action which has the effect of sending acknowledgement of mail having been read without the recipient knowing about it or it may explicitly solicit a response from the recipient and send it either to the sender or to some other recipient. As an extreme action it may even destroy itself (the mail) in order to preserve secrecy.

Our model fits equally well in either a sequential processing environment or a concurrent processing environment. In the case of a sequential processing environment the mail is reduced to a procedure executing at a remote site on local data (local, since the data is sent along with the procedure) under the control of the mail manager. Although similar to remote procedure calls (RPC) it is not the same as RPC. In a sequential processing environment, the mail manager has to act as the intermediary between the recipient and the sender's mail. In a concurrent processing environment, the mail manager only acts as an intermediary between the two processes (the user process and the process corresponding to the received mail) and its job is done once the two processes are informed about each others existence.

Our model differs from the traditional mail system model in two ways. Traditional mail systems consider mail to be a sequence of characters whereas, we consider a mail to be a computational entity. Secondly, in the traditional mail systems, the senders mail becomes recipient property (or data) after it is received by the recipient. Opposed to this, in our model the mail retains its identity on arrival at the receiver and remains a distinct computational entity.

However, we can simulate the behavior of the traditional mail system by coding this action as part of the mail.
6 Research Questions

There are quite a few questions to be answered with respect to our model. The foremost question concerns a formal model for our message system. One approach is to consider the mail as an object and then extend the actor model [1]. The model described by Agha [1] does not include messages as objects.

The other questions relate to implementations of the message system. In the case where the mail system is distributed over a geographic area the underlying computational environment is necessarily loosely coupled. In this case, the computational resources at the different sites may be homogeneous or non-homogeneous. If the computational resources at the different sites are homogeneous then it is possible to send the operations part of the mail in the executable format. However, this raises a problem with respect to the communication of the mail message. The problem is related to the fact that part of the mail is binary (the operations) and part of the mail is text (the data). In a heterogeneous system we have to decide on a methodology to allow operations to execute at a remote site running under a different computational environment. One approach consists of defining a set of primitives which are supported by the mail managers at all the sites and then coding the operations via these primitives (similar to the file transfer protocol primitives). This in turn raises further issues, namely, the constitution of an adequate set of primitives and mechanisms to write user operations using these primitives. Depending on the size of the set of primitives we could either enhance the mail manager to provide a user interface which would convert user actions into primitives or in the case of a large set of primitives we may need to think in terms of defining a new programming language catering to mail oriented applications.

7 Conclusions

We view the study of "message objects" as analogous to the study of object-oriented programming languages. Our intention, however, is to provide an intelligent way of representing messages and communications for office automation purposes. We propose the notion of "active" and "intelligent" message objects. Active and intelligent because of the capabilities with which the messages are endowed but also because of the way the messages interact with the users. This approach opens new possibilities for implementing office procedures and suggests that delegation of responsibility can be performed in a more distributed and natural ways.

We need to understand the properties of a formal model for representing messages as objects before we can effectively build an advance communication management system. We have intuitively described how message objects can be used as an intelligent way or as a framework for an advanced communication system but issues such as formal definition of semantics, completeness of the system, etc., need to be fully investigated so that we can increase our confidence of such communication system.

**Formal Definition of Semantics.** The main goal of a semantic model in this context would be to capture natural structure, and describe the properties and plausible restrictions in a communication system that are physically realizable. A semantic model would formally and mathematically document the structure of the message system at varying levels of detail, would aid users in interpretation of the system, and would allow the proposed advance message system to be specified, developed, and verified in a systematics rather than ad hoc manner.

We are currently investigating various formal models, most notably the actor formalism as defined by Agha [1], as the semantic model for our proposed message management system. The actor model not only captures the abstract power of the object-orientation but provides as well a mathematically precise abstract machine for analysis of asynchronous and concurrent computations based on message passing.

Acknowledgement

The first author wishes to thank Zohreh Talebizadeh for inspiring the idea of "messages as objects." Her work in [20] has also influenced this paper.

References


[4] Church, A.: "The Calculi of Lambda Conve-

sions," Annals of Mathematical Study 6, Princeton

Univ. Press, Princeton, 1941.

[5] Ellis, C., and Naffah, N.: Design of Office In-

formation Systems, Surveys in Computer Science,


vices Bibliography," Technical Report NIC-Biblio-

91901, SRI International, Melno Park, CA (Dec.

1979).


Automation," IEEE Transactions on Communica-


[8] Guttag, J.: "Abstract Data Types and the Devel-

opment of Data Structures," Communications of


uous Functionals," In Formal Description of Pro-

gramming Concepts, E.Neuhold (editor), North-

Holland, 1977.


fice Automation, D. Tsichritzis (editor), Springer-

Verlag, 1985.


Oriented Concepts, Databases, and Applications,

Addison-Wesley, 1989.

[12] MacQueen, D.B.: "Models for Distributed Com-

puting," Rapport de Recherche No. 351, Labora-

toire de Recherche en Informatique et Automatic

e, April 1979.


ing Specification in Office Information Systems," 

ACM Transactions on Office Information System,


tems," Proc. of the Workshop on Applied Com-


[15] Natarasat, O.: "A Survey of Object-Oriented Con-


Achieve Resource Sharing," In Proc. of Spring 

Joint Computer Conference, ACM, pp. 543-549, 

1970.

[17] Saiedian, H., and Unger, B.: "ABSL: An Actor-

Based Specification Language for Office Automat-

ion," Proc. 1990 ACM Computer Science Con-

ference, pp. 252-258, Washington, DC, (Feb. 

1990).


Directions in Object-Oriented Programming, MIT 


and Business Procedures for Office Automation," 

IEEE Transactions of Software Engineering, SE-8 


Organisms: Can the Comparison Improve the 

Logical Modelling of Complex Systems?" Accepted 

by the Fifth International Conference on Symbolic 


ured Messages," IEEE Transactions on Commu-


[22] Vittal, J.: MSG Manual, Bolt Beranek and New-


[23] Vittal, J.: "Active Message Processing: Messages as 

Messengers", In Proceedings of the Interna-

tional Symposium on Computer Message Systems, 

IFIP TC-6, Ottawa, April 1980.