Design principles of an office specification language*

by MICHAEL HAMMER and JAY S. KUNIN

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Office automation, interpreted most generally, is the utilization of technology to improve the productivity and quality of office work. This concept encompasses a wide range of devices, technologies, tools, and systems. One of its most powerful instances is the notion of an automated office information system. This is a software-intensive, computer-based system that seeks to support (and where appropriate, to automate) an entire office procedure, rather than simply to improve the performance of individual office tasks. However, there is a major impediment to the realization of such systems: because of their application-oriented and office-specific character, they are extremely costly to construct. One of the major reasons for this cost is that office systems analysts lack any tools or methodologies to employ in the process of determining and expressing the requirements of an automated office system. An office specification language is used to describe in a natural yet precise fashion the operation of an office system; its use can improve the process of constructing the system in a number of ways. In this paper, we set forth an approach to the design of office specification languages and present an overview of the major concepts in OSL, one such language that we are developing.

OFFICE SYSTEM IMPLEMENTATION

Any system or tool that is sufficiently general to be employed without modification in a wide range of office contexts cannot, by definition, be oriented toward the particular needs of a specific office. Such a system addresses a lowest common denominator of office work; thus, its impact on office work and office productivity will inherently be limited. It is only by taking a holistic approach to the activities in an office, by identifying and understanding the office functions performed and the processes conducted to realize them, and by designing and implementing a system to support them, that major improvements in office productivity will be realized. A key concept here is one of an office procedure, an overall framework that provides an organization and order for the individual activities performed in the office. An automated office information system seeks to improve the execution of office procedures, by improving the performance of specific parts of the procedure and by creating an environment for the integration and control of the procedure as a whole.

An automated office system is an integrated and interconnected collection of components under the supervision of an intelligent control program. These components may be mechanized tools designed to support people in performing unstructured office tasks, or they may be automated subsystems that by means of preprogrammed instructions execute routine and highly structured office tasks. (This distinction between automation and mechanization is fundamental for appreciating the potential of office systems in improving the realization of office work.) By substituting machine labor for human labor where appropriate, and by addressing the entire office rather than just isolated tasks within it, such an office information system represents the paradigm that must be followed to realize the full potential of the new office technology.

However, there is a fundamental problem with this approach to office automation: it is the issue of building office-specific information systems in a cost-effective fashion. No two offices operate in the same way or follow exactly the same procedures; therefore, the paradigm of installing off-the-shelf generic products into an office environment is no longer appropriate when attempting to realize functionally oriented office systems. Instead, a system development effort is required, in which the operations of the office in question are analyzed, its needs assessed, and a custom system designed and implemented for it. The last stage of this process will entail the construction of software that is specific to the particular office in question. This software will embody knowledge of the office's operation; it will automate selected clerical tasks, control the assorted devices employed in the system, and serve as the intelligence that organizes and orders the steps of the office procedure as a whole. This software is clearly specific to the office in question. In other words, custom software must be produced for each office information system.

The difficulty with this approach to office automation is, of course, the fact that it is very expensive to produce office information systems in this way. The process outlined above calls for highly trained personnel (systems analysts and programmers) who must exercise ingenuity in analyzing the

* This research was supported in part by Exxon Enterprises, Inc.
operation of the office in question, defining its needs, and
designing and then implementing a system. Moreover, ex-
perience has repeatedly shown that complex software sys-
tems produced by conventional means tend to be error-
prone, costly to construct, and difficult to change. If we are
to be successful in building and installing custom office sys-
tems on a wide scale, we must seek new means to produce
them.

Let us look more closely at the process of office system
construction, and at the problems with it. It is possible to
identify four stages in this activity: analysis, specification,
design, and implementation. In the first stage, the current
operations of the office are studied and their shortcomings
identified, and the general capabilities of the automated sys-
tem to be built are defined; in the next, precise specifications
of this system are produced. These are the tasks of the office
systems analyst. The programmer then designs the structure
of a system that will meet these specifications and finally
reduces them to code. The major sources of difficulty in this
process lie with the analyst’s activities rather than the pro-
grammer’s. The programmer need only to seek to implement
the analyst’s design; there are few guidelines or principles for him to
employ. One particular problem the analyst faces is that he has no
effective notation or language in which to express himself.
Many errors in software systems arise from the fact that the
original specifications for the system are unclear, incorrect,
or incomplete; this derives from the fact that they are poorly
expressed in a language unsuitable for the purpose. Cur-
cently, an office analyst will use English to describe the cur-
rent operation of an office as well as to specify the desired
functionality of an automated system that is to be built.
Although rich and expressive, English, like all natural lan-
guages, is imprecise and ambiguous and consequently not
useful for the accurate specification of systems. Specifi-
cations must bridge the gap between the analyst (who is ori-
ented toward application constructs and office needs) and the
programmer (who is concerned with the design and im-
plementation of software systems). This gap is the breeding
ground for ineffective communications, expense, and error.

OFFICE SPECIFICATION LANGUAGES

We believe that many of the problems discussed above
can be significantly mitigated by providing the analyst with
a problem-oriented office specification language. This is a
formal language for describing in high-level and machine-
independent terms the operation of an office system (either
manual or automated). It may be thought of as a notation
in terms of which an office system analyst can express him-
self, both for describing an existing office operation and for
specifying the operation of an automated system to be built
by the programmer. A specification language is a formal lan-
guage, with rigorously defined syntax and semantics. Thus,
any description expressed in it is unambiguous and open to
a single interpretation. Furthermore, the primitives of a high-
level language are based on the natural structures and vo-
cabulary of office work so that the language user can express
himself in terms natural to the problem domain. Such a lan-
guage can serve as an effective means for specifying in a
precise, natural, and understandable way the operation of
an office system.

We envision a variety of potential uses for such a language.
The principal one is as a communications mechanism be-
tween office systems analyst and programmer. Because of
its formality and precision, specifications expressed in the
language can be clearly understood and interpreted by the
programmer who must use them as the basis for his system
implementation effort. The use of this language will enable
an office systems analyst to describe more precisely to a
programmer the system that is to be constructed; this im-
proved communication can have a major and positive impact
on the systems thus produced, improving their quality and
lowering their cost. The use of such a language facilitates
the jobs of both the analyst and the programmer. Because
of the high level of the language, the analyst will be able to
readily express himself in terms familiar to him while sup-
pressing irrelevant detail. Second, the language can impose
a structure on the entire process of office analysis and system
specification. By providing the analyst with high-level primi-
tives in terms of which he is to express a system, a speci-
fication language effectively gives him a set of templates
with which he is encouraged to analyze office operations.
Thus, the analyst is presented not just with a set of discon-
ected language features but with an approach to their em-
ployment, a perspective on office operation that provides
a conceptual framework in terms of which to analyze and
describe office operations. Finally, there are several uses of
such a language that are not directly related to the process
of constructing automated office systems. A formal speci-
fication language for office procedures can serve as a very
effective mechanism for expressing precise and complete
descriptions of the existing manual office operation. In cur-
rent practice, English is the language employed in systems
and procedures manuals; however, as is well-known, these
manuals are usually incomplete, difficult to read, and ob-
solote. Well organized and precise specifications in a high-
level language can be used as a reference for office workers
in many office environments. Related uses are for the train-
ing of new employees, and for the recording of organizational
history in a way that survives the coming and going of in-
dividual office personnel. The formal specifications of an
office procedure can also be subjected to various analytic
techniques in an effort to identify bottlenecks and problem
areas in its operation; this can highlight those areas of the
procedure most in need of rationalization and redesign,
whether or not in the context of an automation effort.

Obviously, the mode in which an office specification lan-
guage is used depends in part on the application for which
it is being employed. However, in general, we expect that
specifications will be written by a trained office systems analyst who has been instructed in the use of the language. This person will not necessarily be a computer expert; he may be a manager, a staff professional, a secretarial or clerical worker, or a specialist dedicated to this task. He must possess two important skills: a deep understanding of office work, and an ability to analyze and describe office operation in a systematic fashion.

We believe that the use of the language will be ongoing but intermittent. That is, at some point the initial description of an existing or proposed office system will be expressed in the language, and on a regular basis this description will be updated to reflect changing circumstances and evolving needs. However, we do not believe that specifications will be modified on an ad hoc basis by individual office workers. While there will be few writers of specifications, we expect there to be a large population of people who will want to read specifications expressed in the language. Readers of office specifications will include office workers who will consult them in order to determine aspects of procedures with which they are not familiar; office trainees in the process of learning the office operation; office managers seeking ways in which the operations of the office can be improved; and programmers who will be called upon to translate such specifications into operating programs.

As suggested above, we believe that this language will be used both for prescriptive and descriptive purposes; that is, to describe an existing office operation as well as a new and proposed one. In fact, such uses are often demanded in the context of an evolving office system. An analyst must first construct a description of the system as it is currently configured and use that as a basis for developing specifications of a new and improved system. It is rarely feasible to institute a revolutionary change in the process of an automation effort and to dramatically restructure an entire office operation; rather, the new office system must evolve from the old one. Consequently, at some suitable level of abstraction, the specifications of the new system should be virtually identical to those of the old one. It is only at the level of mechanism and implementation that the two become distinguishable. Thus, it is appropriate that the specification language be multi-tiered, with the topmost level expressing the implementation-independent structure of the office and only the more detailed level serving to identify the particular way in which the general structure is being instantiated.

**AN APPROACH TO SPECIFICATION LANGUAGE DESIGN**

We are engaged in an ongoing effort to develop such an office specification language (known as OSL) for use in analyzing, describing, and implementing office systems. Based on the foregoing perspective on the use and utility of such a language we have developed the following approach and design criteria that we are employing in this effort.

1. The language must be formal and well defined; that is, it will have a limited vocabulary of constructs that can be combined only in specific ways. As a result, it will be possible to determine in an automatic fashion whether a particular specification in the language is legal and meaningful. Furthermore, any legal specification will admit of only one interpretation. These properties of formally defined languages avoid many of the difficulties associated with English and other natural languages.

2. The language must be highly readable; it should be possible for an individual with a very small amount of training to be able to read and understand specifications expressed in OSL. This criterion is motivated by the fact that there will be a large readership for OSL specifications, most of whom will not be specialists in the language. There are several consequences of this requirement. First, it dictates that the constructs of OSL be natural and problem-oriented rather than general and abstract. That is, the dictions of the language should reflect the natural semantics of offices and office work; the language primitives should directly correspond to office activities and structures so that the description of an office procedure will be couched in terms meaningful to those familiar with office work. Furthermore, every specification expressed in the language should have a manifest and understandable overall organization and structure. That is, not only should individual atoms of the language be easy to comprehend, but a description of an office procedure as a whole should be organized in a way that enables people to comprehend it easily.

3. The language should support the process of writing descriptions by incorporating a standard and natural logical structure for office specifications. We believe that the way to aid someone seeking to write specifications in a language is not by providing him with a minimal set of general and flexible linguistic features. While it may be easy to learn the meaning and capability of each one of these constructs, the entire burden of combining them into a complete specification is then thrust upon the user. A language that is easy to learn is often difficult to use for all but the most trivial of applications. Consequently, we think it appropriate that OSL possess a rather more complex and intricate inherent structure, which may require more effort to learn but which should greatly enhance its usability. In other words, the user of OSL will start out with a preconceived notion of the general structure of the specifications he will produce; his task is to match the particulars of the office in question to the canonical structure. As a consequence, associated with OSL will be a methodology for conducting office analyses and writing specifications, which is based on the same conceptualization of office work embedded in the language.

4. The language should be high-level and nonprocedural. The specification of an office procedure in OSL will be expressed at a level of abstraction corresponding to
5. Specifications expressed in the language must be modifiable. Office procedures are highly dynamic; they continually evolve to meet unanticipated situations and new requirements. The specifications for an office system must consequently evolve in an incremental fashion to reflect these new developments; if they are not readily modifiable, they will inevitably become obsolete and unused. Moreover, modular and modifiable specifications can result in more maintainable software systems. If the software system reflects the structure of the specifications, then as the specifications change, the implementation can often be modified in corresponding and limited ways.

The overall goal of our design effort is to develop a language in which office systems analysts can readily construct highly readable specifications that are clear, unambiguous and natural descriptions of office procedures. These specifications should uncover and highlight the basic structure of the procedure rather than focus on the details involved in its implementation. Our approach is a functional one; that is, we do not find it feasible or even desirable to attempt to capture in a specification all of the mechanisms associated with an office procedure. First, any such “complete” specification will be overwhelming in its size and complexity. Second, it is unlikely that an office system analyst (in any finite amount of time) will be able to uncover all possible variations of the procedure. And, third, the implementation details of an office procedure continually evolve as office workers develop new techniques to solve old problems or face previously unencountered difficulties. Consequently, we have not sought to achieve any elusive “completeness.” Instead, a description couched in OSL will focus on the purpose of the procedure, rather than on its mechanics. This is accomplished by including in the language primitives that express the goals of office activities in application terms. In order to achieve this end, it is necessary to sacrifice completeness in another way as well. We do not expect that OSL will be appropriate for describing all conceivable office procedures. OSL embodies a particular perspective and approach to office work and its description which, we believe, matches a large number of office procedures, although certainly not all of them. Our goal is to make OSL extremely usable for a large class of applications, rather than minimally adequate for the universal class of applications. We have sought to optimize the design so that what OSL does, it does very well; we are devotees of the 80/20 rule. Whether or not we achieve this goal and whether or not the class of applications for which OSL is appropriate will be large enough to justify this decision, only extensive experience with the language will answer.

The fundamental premise underlying the design of OSL is that there is a high degree of commonality of structure and activity among procedures in different offices. In other words, we believe that there are fundamental semantic structures in the office application domain that recur in many different contexts. This commonality can be exploited by identifying the structures that are repeatedly used in natural descriptions of different office procedures and embedding them in a formal language. The user of the language will then find that it provides him with just those problem-oriented features that he wishes to use; he will not have to build up a description of an office procedure from lower-level and more general constructs. Consequently, the design of an office specification language must be based on an extensive familiarity with the application environment.

The first (and an ongoing) aspect of the OSL design effort has been to conduct many case studies of office procedures in different environments. We have conducted analyses of a large number of operational and administrative offices, of several different kinds, within a variety of organizations. It is by analyzing and abstracting from these descriptions that we have identified the fundamental constructs of OSL. Based on these analyses, we have concluded that most office procedures are fundamentally simple processes that are often obscured by implementation details and disorganized exception handling. However, when it is eventually uncovered, the basic structure of the office procedure is often relatively easy to comprehend and describe, given the appropriate set of primitives. The “complexity” of office procedures is often an artifact; the goal of OSL is to manage and even avoid this complexity.

The following summarizes the principal characteristics of OSL.

1. A specification expressed in OSL takes a holistic view of office activities; it is not based on a description of the processing performed on individual forms passing through an office nor around the activities of individual office workers. Rather, it expresses an integrated view of the office activities as a whole, with the focus on the end being achieved rather than the means being employed to achieve it.

2. OSL descriptions are expressed in terms of application-oriented constructs, eliminating as far as possible any detail that is not germane to the application itself but that results only from the fact that the specification is being expressed in a formal language.

3. OSL specifications are highly structured in a canonical way. OSL imposes a uniform format on the description of every office procedure. Furthermore, this description is modular, so that it is possible to develop an understanding of individual parts of it and to comprehend its overall structure without working through all the details. This modularity is accomplished in two ways. First, the language employs techniques of successive refinement so that the procedure can be expressed and understood at multiple levels of abstraction. Second, because of their importance in office procedures, the descriptions of exceptions and special cases are not incorporated directly into the main line
of the procedure, but are attached to it in specific and predetermined ways.

4. OSL embodies the most common specialized constructs used to describe office procedures. While this does lead to growth in the number of language features, with some attendant increase in its complexity, it also leads to shorter and more understandable specifications. The intent of the procedure is evident from its surface, since it is being expressed directly rather than coded in terms of general and abstract facilities.

5. OSL makes extensive use of declarative specification techniques. That is, as much knowledge of the procedure and its operation as possible is embedded not in a description of activities to be performed, but as constraints and restrictions on the data values or documents associated with the procedure. This leads to greatly simplified procedural descriptions as well as specifications that are easier to change (because of the locality of this information).

Below we shall see how these general principles and criteria have been addressed by the current version of the language.

AN OVERVIEW OF OSL

The major premises discussed above have had a major influence on the development of OSL and are reflected in its philosophy and features. A holistic view of office specification is central to the structure of the language. While OSL recognizes the importance of forms and people as individual units of office activity, it does not structure a procedure description around them; its orientation is toward the objects in an office. Objects in this context are the entities that are the focus of office activities and that form the basis for a description of office functions; the office as a whole is described in terms of the evolving history of its objects. OSL provides canonical high-level office-oriented constructs for the specification of both data and control structures. Such constructs provide a framework for the organization and presentation of a specification and also act as a guide to the analyst in structuring his task. This framework in turn provides for the readability and naturalness of expression necessary for using OSL in documentation and training. Finally, OSL provides a built-in structure for all specifications; the office description as a whole has a standard format, and each of its components can be decomposed in a uniform way.

We shall now describe in some detail the structure of the initial version of OSL. In this discussion, we will employ as an example the Office of Sponsored Programs (OSP) at MIT, whose major functions are to expedite the submission of research proposals and the negotiation of contracts, and to monitor the resulting grants and contracts to ensure compliance with internal policies and contractual requirements.

An office specification in OSL consists of two major components: a description of the application domain with which the office is concerned, and specifications of the procedures performed in the office. The former provides a context for the description of the procedures. It effectively expresses a model of the world of the office; it describes the objects on which the office is focused, the organizational context of the office, the documents and forms that the office processes, and the information that it needs to utilize. In the case of the OSP, this contextual information describes a world consisting of proposals, contracts, funding agencies, researchers, laboratory directors, and the like. The description is couched in terms of a variant of the SDM, a data modeling mechanism originally developed for describing the information content of databases. The key feature of the SDM is that it models an application rather than data; thus the specification includes a direct description of the office and its environment. This enables the specification to distinguish substance from artifact; a procedure can access information it requires by directly referring to the appropriate attribute of an entity, without caring whether that information is captured on a form, in a database, or elsewhere. This "schema" thus naturally expresses the static semantics of the office in terms with which a reader is likely to be familiar.

The description of the office environment is expressed in terms of entities and their attributes, inter-entity relationships, and entity collections. Associated with the description of an office entity is the definition of those documents related to it (for example, a proposal document is associated with each proposal entity), as well as constraints on the attributes of the entity and on its processing (for example, that the principal investigator of a contract must be a faculty member or that if human subjects are to be used in the research then approval must be obtained from an appropriate university committee). By associating constraints with objects and documents, the description of context becomes more meaningful and the specification of the procedures becomes simpler and more modular. Specialized entity and relationship types (such as people, agreements, schedules, logs, supervision, and the like) are provided, since they recur frequently and they possess special semantics. Included in this environmental specification is a description of the offices in the organization and the lines of communication and authority that connect them. Such an organizational description is particularly valuable when a function is realized by means of related procedures executed in different offices. The local office context describes the people in the office, their roles, responsibilities and authority, as well as the files maintained in the office. The environmental description also identifies the primary objects of the office. These are the entities that are the major focus of the office activities and around which the descriptions of the procedures are organized. In the OSP, the primary objects are proposals and contracts.

The dynamics of the office are captured in the specification of its procedures, the activities it performs in the described context. Just as an appropriately-designed data model can serve as the basis for natural descriptions of the environment, so too a simple but powerful model of office activities can be applied to procedural specification. To this
end, OSL incorporates a canonical set of structures for process description that are based on three concepts: an orientation around objects; hierarchically structured and modular descriptions; and an emphasis on the identification of exceptions.

Fundamentally, every OSL procedure is concerned with processing and/or managing a primary object. Based upon such an object orientation, we find that a large number of office activities can be described in terms of a fairly simple three-stage model representing the “life cycle” of an object: an initiating procedure, an administrative procedure and a terminating procedure. The primary object has three versions, which correspond to the three stages of the procedure.

An initiating procedure manages an initiator object. This is an “active” procedure; it “pushes” the initiator through a series of operations to an end point, at which time the administrative procedure is triggered. In the OSP, the initiator is a contract proposal, and the initiating procedure is concerned with obtaining institutional approval for the proposal and negotiation with the proposed sponsor. At the conclusion of the initiating procedure (which may include several iterations of the proposal submission process), the proposal is either rejected, terminating the overall procedure, or accepted, invoking an administrative procedure. An administrative procedure manages a (set of) resource objects. It is concerned with maintaining the status of the resource and verifying and recording all activities that are applied to it. In the OSP, the resource is a contract; the process involves assuring that the spending restrictions and reporting requirements of the contract are honored.

The final stage of the “life cycle” of an object is termination. When the resource is no longer of interest, the administrative process triggers a termination procedure, which produces an archive object. This is another “active” procedure, although in many cases it is relatively uncomplicated. An archive object simply represents any information that must be available after the termination of the procedure and destruction of the resource.

We have thus defined two classes of procedures: administrative and active. An administrative procedure is specified by means of a formatted description that identifies periodic inputs and outputs and how they are to be handled (progress reports and accounting reports in OSP) and nonperiodic, but expected, events, together with the processing required to handle them (e.g., purchase authorizations).

Active procedures are specified somewhat differently. An active procedure can be viewed as the application of a set of activities (“verbs”) to an object by a responsible agent (“subject”). The control structure for specifying the order in which activities are to be applied is formalized in the following manner:

The procedure is described in terms of states, events, and activities. States are stages in the execution of the procedure at which no further processing can be done until the occurrence of some event. An event is an autonomous occurrence that is beyond the control of the agent responsible for the procedure (e.g., the receipt of a document, the arrival of a specific date and time). When the procedure is in a given state, it is waiting for the occurrence of one of a designated set of events; when one of these events occurs, a corresponding activity is executed, at the conclusion of which the procedure is left in some other state. A state machine-like formalism can be used to express these relationships and determine the overall control structure of the procedure. For example, one state of the OSP initiating procedure corresponds to a situation in which a proposal has been sent to the legal office for review; when an appropriate event (receipt of a response from the legal office) occurs, the proposal can be further processed by OSP preparatory to its submission to the proposed sponsor.

Activities are the specific actions performed in the course of a procedure; the description of an activity, like that of a procedure as a whole, is hierarchical and modular. At the top level of specification, activities are specified with a minimum of detail; this is provided in the lower levels of description. This hierarchical structure supports the modification of activity descriptions; it also allows for both descriptive and prescriptive (normative) specifications of any given office situation. We anticipate that the top-level specification will express the goal of the activity; the detail may represent either the results of the analysis (description) or the specification of an implementation (prescription).

At the base of the activity hierarchy is a set of activity primitives, each of which is a fundamental office operation. For example, the primitive “select” is used to describe a decision in which a subset of available resources will be chosen. (The “allocate” primitive indicates the opposite situation.) Note that even at the primitive level, we seek to describe function, rather than implementation. Thus, “select” may be performed in various ways: we may have an algorithm for implementing it (e.g., to select the highest-scoring applicant), or it may be inherently judgmental (e.g., to select a site for a new plant). This flexibility in describing activities is desirable in a tool for analysis and documentation.

One of the key aspects of our formulation is its approach to exception handling. As we have discussed, special cases and exceptions are often the source of the perceived complexity of many office procedures. By organizing the description of exceptions, we provide a means of making the overall specification more organized and readable.

We take a hierarchical approach to the specification of exceptions. Each level of procedure and activity description has an associated list of exceptions. These are classified in terms of the nature of the events that give rise to them: instances include violation of timing constraints, invalid data values, unavailable personnel, and activity-specific errors (e.g., an inadequate set of resources from which to make a stipulated selection). The responses to the exceptions are also frequently drawn from a canonical set. The descriptions of the exceptional situations and the responses to them are separated from the mainline procedure to enhance the latter’s readability.

RELATED WORK

We note that our usage of the concept of a specification language differs from that typically employed in the computer science literature. The common usage refers to a gen-
eral-purpose facility for describing the behavior of individual modules of an arbitrary large software system. By contrast, our perspective is domain-specific and implementation-independent. However, some earlier work has been done in areas related to our effort, generally in the context of business information system design. The Time Automated Grid and Accurately Defined Systems languages are designed for describing file processing applications; the latter is primarily a documentation tool. The Business Definition Language is also aimed at highly-structured data processing tasks. The Problem Specification Language is a more general language for defining information system requirements. Although a number of ideas useful for the specification of office systems can be derived from these languages, it is clear that their scope is inadequate for the flexible, interactive, and semistructured nature of the systems that we are addressing.

Recently, several attempts have been made to design languages specifically for the office domain. Barber and Hewitt are using a form of the Actor formalism to specify the activities of office workers and the communications among them, primarily in an effort to find means of symbolically proving, simulating and modifying office procedures. IBM's System for Business Automation Programming Language is based upon the Query-by-example relational database query language. The user programs in a forms-oriented graphical environment; primitives in the language include database access, forms editing and control, and similar functions. The Xerox PARC Officetalk system uses a similar forms-oriented interface and programming-by-example paradigm, although it also allows the writing of procedural descriptions of forms processing with a small set of filing, communications, and forms editing primitives.

A major problem with all these languages is that they do not deal directly with office function. They are oriented toward the tasks of individual office workers; the focus is on current task structure, rather than fundamental function. As a result, the utility of these languages for high-level specification is limited.

The Office Procedure Specification Language does deal more directly with office function. It primarily allows the description of documents and communications patterns. Primitives are at the level of document definition and movement; more complex process is expressed by programs written in a general-purpose programming language. The structure and syntax of the language are tightly coupled to an augmented Petri Net formalism, a general representation scheme for asynchronous, concurrent processes. The major shortcoming of this approach, which it shares with all of the above office description languages, is that it lacks any constructs at a level higher than "send message" or "file document." Nor do any of these languages result in highly structured or readable specifications.

**SUMMARY**

We have presented the basic concepts of an office specification language and identified the principles on which the design of the OSL design effort is continuing. A first version of the language has been specified, and it is being applied to a number of test cases. Experience with the use of the language will indicate needed changes; we expect that this iterative approach will lead to a highly effective language. We have also begun exploring the design of an OSL processing system, which would seek to automatically generate an office information system from its OSL description.

**ACKNOWLEDGMENTS**

We would like to acknowledge the valuable suggestions of our colleagues Marvin Sirbu and Sandor Schochet.

**REFERENCES**
