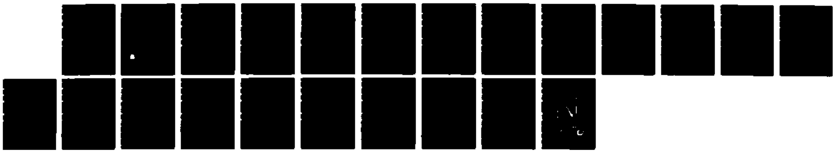
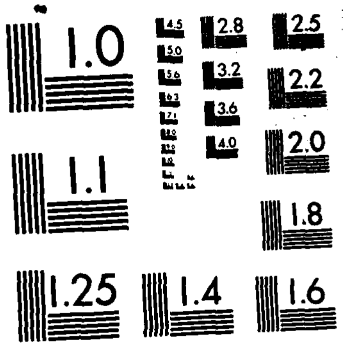
AD-A164 632 THE ZOG APPROACH TO DATABASE MANAGEMENT(U) 1/1
CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER
SCIENCE R M AKSCYN ET AL. MAR 84 CMU-CS-84-128
UNCLASSIFIED N00014-76-C-0874 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

CMU-CS-84-128

0

AD-A164 632

The ZOG Approach to Database Management

Robert M. Akscyn and Donald L. McCracken

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

March 1984

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

86 2 19 026

CMU-CS-84-128

The ZOG Approach to Database Management

Robert M. Akscyn and Donald L. McCracken

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

March 1984

Abstract. ZOG is a general-purpose human-computer interface system that combines the features of a database system, a word processing system, and an operating system shell. The primary features of ZOG are: (1) an emphasis on menu-selection as the primary interface mode; (2) the use of the selection process for navigation in the database, editing the content and structure of the database, and interaction with programs; (3) an architecture that supports the implementation and growth of very large, distributed databases; and (4) rapid system response. A distributed MIS, based on the ZOG concept, was developed by Carnegie-Mellon University for the USS CARL VINSON, a nuclear-powered aircraft carrier, in cooperation with the ship's crew. This system is a distributed database system implemented on a network of high-powered personal computers (PERQs). This paper focuses on ZOG as a database management system. Using a set of common database problems as a framework, the ZOG approach to database management is discussed and compared with conventional approaches.

This work was supported by the Office of Naval Research under contract N00014-76-0874. It was also partially supported by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the Defense Advanced Research Projects Agency, or the U.S. Government.

This paper is to appear in the Proceedings of the 1984 Trends and Applications Conference: Making Database Work, Gaithersburg, MD, 23-24 May 84.

Table of Contents

1. Introduction

ZOG is a general-purpose human-computer interface system based on very rapid response to user selections which enable the user to browse among and edit screen-oriented chunks of a large network-structured database [1, 2]. ZOG was developed in the Computer Science Department of Carnegie-Mellon University beginning in 1975. During the four-year period from February, 1980 to February, 1984, the ZOG Group at CMU developed and installed a computer-assisted management system for the USS CARL VINSON, a nuclear-powered aircraft carrier [3]. This system, developed jointly with the VINSON crew, consists of 28 high-powered personal computers (PERQs) interconnected via a 10 mHz local-area network (Ethernet). A network-structured database is distributed over these machines in a manner transparent to both users and applications programs.

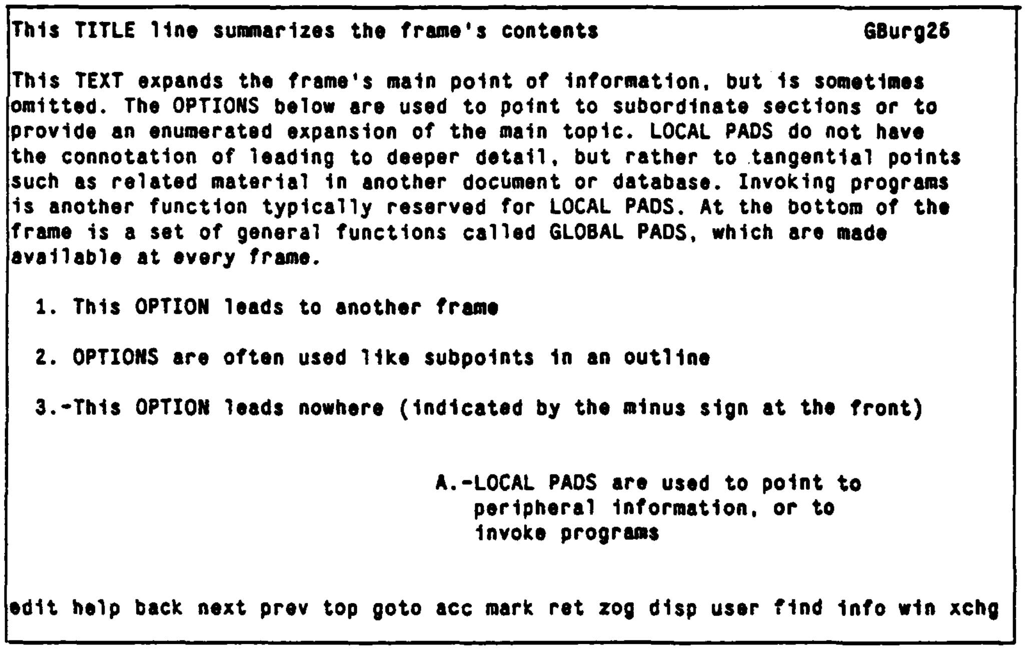
This paper presents some of the features of the ZOG system that may be of interest to designers of database systems, plus some recommendations based on our experience. We begin in the following section with a brief introduction to ZOG, from a database perspective (though ZOG is actually a quite general system and can be viewed in other ways as well, e.g., as an operating system shell). Then in Section 3 we present what we have learned about ZOG as a database, using as a framework a set of common database problems that ZOG addresses. In Section 4 we describe some extensions to ZOG to enhance its usefulness as a database system. In Section 5 we conclude with some general recommendations for database design which summarize the spirit of the ZOG approach.

2. ZOG from a Database Perspective

2.1. Viewing ZOG as a database management system

Although ZOG was originally conceptualized as an "interface", it is not normally presented to users in terms that are familiar to the database community. For example, terms such as "file", "record", and "data dictionary" are not used to describe ZOG. In part, this has happened because the architecture of ZOG has largely been shaped by the needs of humans, whereas database architectures have been shaped primarily by the needs of programs. In this section, we attempt to recast ZOG in database terms so that the correspondence with traditional database systems can be seen more clearly.

ZOG can be viewed as a database management system. A ZOG database, called a *net*, is most similar to databases that use the network data model--though there are no "owner/member" constraints whatsoever. A node in a ZOG net, called a *frame*, can be thought of as a set of records of several different types: frame title, frame text, option, local pad, and global pad. Although there are some minor exceptions, each of these record types contains the following field-like information: a text string which is displayed, an optional pointer to another frame, an optional *action* (written in a ZOG-specific programming language), and display position. A frame consists of one frame title, one frame text, an arbitrary number of options (usually less than 10), an arbitrary number of local pads, and a fixed set of global pads. The semantic distinctions between these types is illustrated in Figure 2-1 below.



```
This TITLE line summarizes the frame's contents                  GBurg26

This TEXT expands the frame's main point of information, but is sometimes
omitted. The OPTIONS below are used to point to subordinate sections or to
provide an enumerated expansion of the main topic. LOCAL PADS do not have
the connotation of leading to deeper detail, but rather to tangential points
such as related material in another document or database. Invoking programs
is another function typically reserved for LOCAL PADS. At the bottom of the
frame is a set of general functions called GLOBAL PADS, which are made
available at every frame.

  1. This OPTION leads to another frame

  2. OPTIONS are often used like subpoints in an outline

  3.-This OPTION leads nowhere (indicated by the minus sign at the front)


                              A.-LOCAL PADS are used to point to
                                 peripheral information, or to
                                 invoke programs


edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 2-1: A self-describing ZOG frame

The frame is the basic building block for representing information in a ZOG net. A typical ZOG net contains tens of thousands of interconnected frames. The default mode of interaction with ZOG is called *navigation*, in which the user selects one of the items from the frame (via single keystrokes or the mouse). The system responds by displaying the frame that the selected item "points" to, typically in under one second. The user can then make another selection. Most of the items on a frame lead to other frames, but some have actions that perform some utility function or run some program. The user can enter the ZOG editor at any frame to make changes or additions. ZOG nets are typically treated by users as communal property, being developed incrementally (one frame at a time) by the users themselves, in a highly decentralized manner (much like a colony of bees building a hive).

The two-dimensional structure of a ZOG frame thus provides the user with a default view of the contents of the database. In practice, this model of a frame has been sufficient to represent both content information and structural information. In some systems, such as the typical operating system, these functions are provided by separate entities, each with its own set of commands (e.g., files for content, and directories for structure, as in the Unix operating system). The model of a frame, the rapid response to navigation commands, and immediate access to the editor make both the content and structure of the database directly viewable and directly manipulable. The importance of the property of *direct manipulation* in a human-computer interface is discussed by Shneiderman [4].

2.2. Implementation of the distributed database

A principal objective of our project with the VINSON was the transfer of ZOG from a mainframe version (written in an experimental language) to a distributed version for a network of personal computers. The current implementation, written in Pascal, runs aboard the VINSON on a network of 28 PERQ computers interconnected by a 10 mHz Ethernet. The system is designed so that the database may be distributed transparently across any subset of machines on the network (a single Ethernet can accommodate up to 1024 machines). Neither users nor programs need to know anything about the names of files or their location.

The local disk for each machine houses some set of *subnets*. A subnet is a logical collection of frames, each bearing a unique identifier composed of the name of the subnet plus some frame number. Typically, users create new subnets whenever they begin building a logically distinct entity, such as a new document or program module. The machine location of each subnet is stored in a special index table maintained by the system on each machine, as well as on a "master" machine. When a frame is requested, the local copy of this subnet index is consulted first to learn on what machine the subnet is located. If this information is not present in the local copy, the master machine is then queried. If the frame is on another machine, a request for the frame is sent over the Ethernet to that machine, utilizing a special high-level protocol which coordinates the processes of creating, reading and modifying frames between machines. Accessing and displaying a remotely located frame takes an average of 1.2 seconds, compared with 0.6 seconds for local frames.

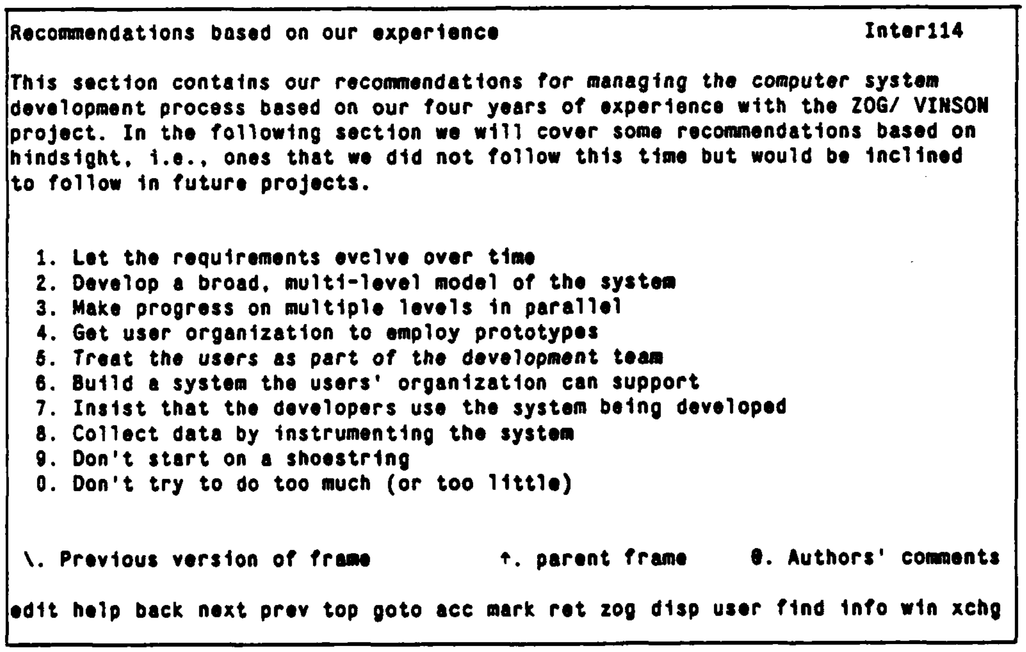
2.3. Examples of ZOG as a database

This section illustrates how ZOG has been used as a database to support applications areas such as the following:

- Documentation development and management
- Software development and management
- Electronic communication
- Project management
- Other applications

2.3.1. Documentation development and management

All of our project documents, including this document, have been developed within ZOG nets. An example frame, taken from one of our recent papers, is shown in Figure 2-2 below. We have found that using ZOG as an authoring environment provides a number of advantages over systems that are strongly linear in structure. The document is continuously maintained on-line in a formatted form, and is quite readable since representing the document as a structured collection of frames obviates the need for embedded structural formatting commands. High quality hardcopy is produced from the net representation by a program that maps the net structure into a file with the appropriate commands for a document formatting system (we use Scribe). ZOG can also be used to represent a documentation database, with indexes and inter-document cross-references, for thousands of documents. In addition, this method of representing documents makes it convenient to associate with documents other "scaffolding" information, such as authors' notes, reviewers' comments, previous drafts, etc. Because locking in ZOG is done at the individual frame level, multiple authors can easily work on the same document simultaneously. On a number of occasions, personnel located in different parts of the country have concurrently developed a document on our ZOG Vax.

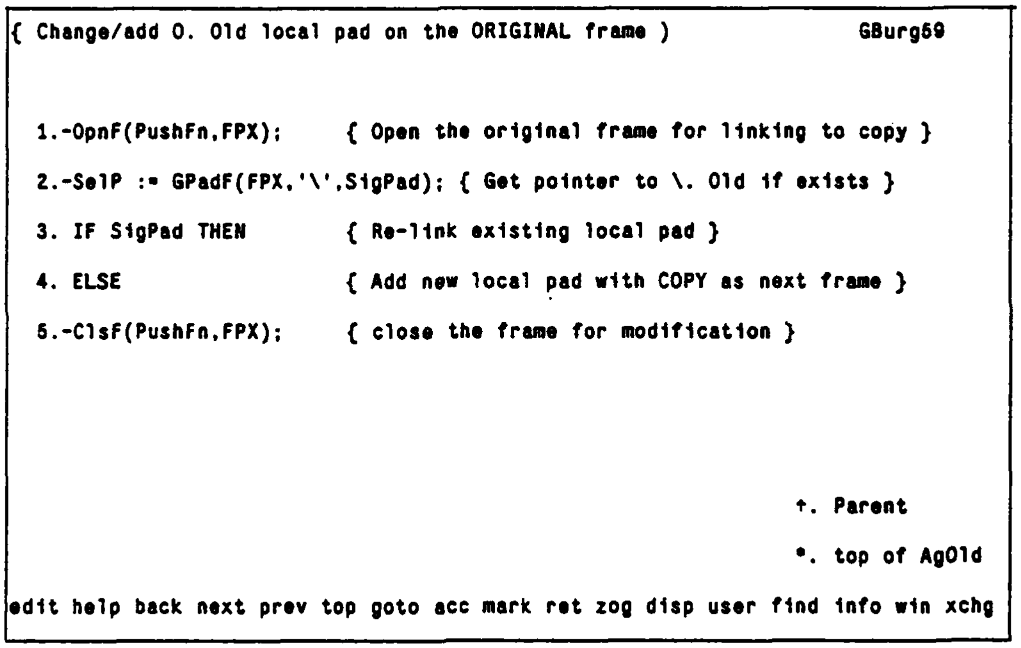


```
Recommendations based on our experience                          Inter114

This section contains our recommendations for managing the computer system
development process based on our four years of experience with the ZOG/ VINSON
project. In the following section we will cover some recommendations based on
hindsight, i.e., ones that we did not follow this time but would be inclined
to follow in future projects.

  1. Let the requirements evolve over time
  2. Develop a broad, multi-level model of the system
  3. Make progress on multiple levels in parallel
  4. Get user organization to employ prototypes
  5. Treat the users as part of the development team
  6. Build a system the users' organization can support
  7. Insist that the developers use the system being developed
  8. Collect data by instrumenting the system
  9. Don't start on a shoestring
  0. Don't try to do too much (or too little)

 \. Previous version of frame        ↑. parent frame     @. Authors' comments

edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 2-2: An example document frame

2.3.2. Software development and management

We have used ZOG as a software management database, by representing Pascal code in frames, along with other related data such as change logs, bug reports, old versions, design notes, and user guides. ZOG's style of top-down, stagewise refinement of hierarchical structures is well suited for structured software development as advocated in software engineering circles. A special program can be invoked from within ZOG to translate the code in the frames into a compilable version in a file. Figure 2-3 shows a sample frame of Pascal code. Note that each statement is a separate option, and that options 3 and 4 both lead to other frames, which contain more statements as in a BEGIN-END block. The structural assistance provided by ZOG can be viewed as a mild form of the syntax-directed approach to programming, with a bias toward "programming-in-the-large".

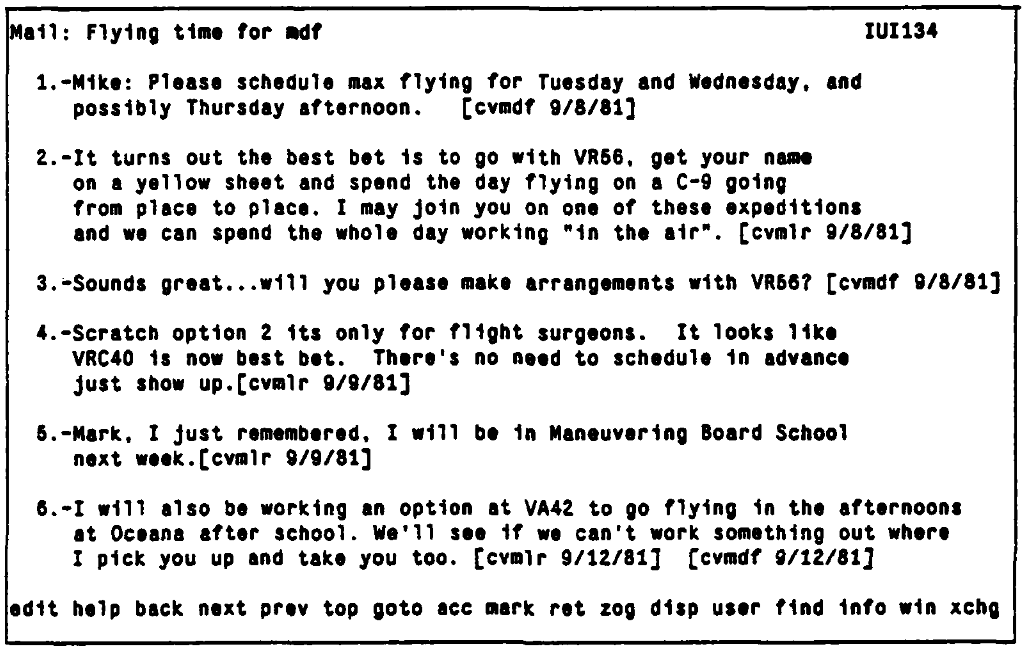


```
{ Change/add 0. Old local pad on the ORIGINAL frame )           GBurg69


  1.-OpnF(PushFn,FPX);      { Open the original frame for linking to copy }

  2.-SelP := GPadF(FPX,'\',SigPad); { Get pointer to \. Old if exists }

  3. IF SigPad THEN         { Re-link existing local pad }

  4. ELSE                   { Add new local pad with COPY as next frame }

  5.-ClsF(PushFn,FPX);      { close the frame for modification }




                                                          ↑. Parent

                                                          *. top of AgOld

edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 2-3: Example Pascal code frame

2.3.3. Electronic communication

We have also used ZOG extensively to implement forms of electronic communication, such as electronic mail, bulletin boards, and teleconferencing. ZOG reverses the mode of operation that is traditionally used to implement these functions. Authors don't "send" messages--they simply go to appropriate locations in the ZOG net and add them. This method permits the global structure of the information across many messages to be developed over time and preserved. Readers can find desired information by navigating through the net. There is also a "find" mechanism, which can be used to locate all the frames changed since an individual's last search. Using this approach to electronic communication, an organization can evolve a well-structured form of corporate memory useful to both old and new members. Figure 2-4 shows an actual frame used by two Navy officers (one stationed with us in Pittsburgh and the other at Newport News, Va.) to conduct a dialogue over a period of several days. Note how each increment benefits from the preservation of the evolving context.

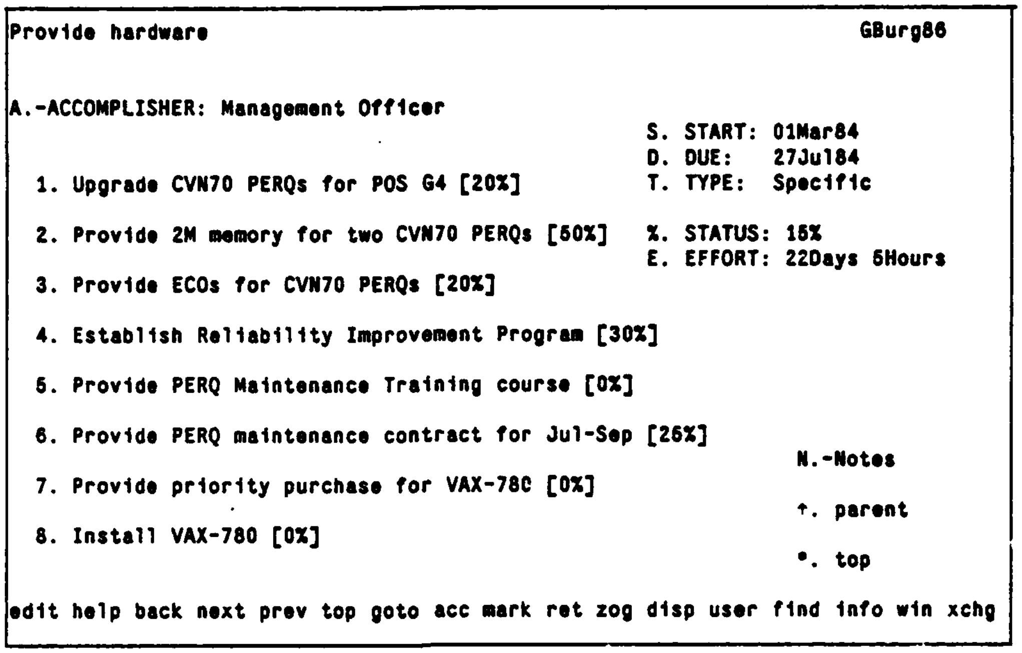


```
Mail: Flying time for mdf                                    IUI134

  1.-Mike: Please schedule max flying for Tuesday and Wednesday, and
     possibly Thursday afternoon.   [cvmdf 9/8/81]

  2.-It turns out the best bet is to go with VR56, get your name
     on a yellow sheet and spend the day flying on a C-9 going
     from place to place. I may join you on one of these expeditions
     and we can spend the whole day working "in the air". [cvmlr 9/8/81]

  3.-Sounds great...will you please make arrangements with VR56? [cvmdf 9/8/81]

  4.-Scratch option 2 its only for flight surgeons.  It looks like
     VRC40 is now best bet.  There's no need to schedule in advance
     just show up.[cvmlr 9/9/81]

  5.-Mark, I just remembered, I will be in Maneuvering Board School
     next week.[cvmlr 9/9/81]

  6.-I will also be working an option at VA42 to go flying in the afternoons
     at Oceana after school. We'll see if we can't work something out where
     I pick you up and take you too. [cvmlr 9/12/81]  [cvmdf 9/12/81]

edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 2-4: Example mail frame

2.3.4. Project management

On a higher plane, we have used ZOG for project management. In addition to the functional areas we have discussed above, we used ZOG tp develop multi-level task structures which could be used not only for planning, but for implementing and evaluating as well. One of the major applications developed for the USS CARL VINSON consists of a task management system based on the ship's organization manual. Figure 2-5 below illustrates this application with a task frame for the Management Officer on the VINSON.



```
Provide hardware                                                    GBurg86

A.-ACCOMPLISHER: Management Officer
                                                   S. START: 01Mar84
                                                   D. DUE:   27Jul84
  1. Upgrade CVN70 PERQs for POS G4 [20%]          T. TYPE:  Specific

  2. Provide 2M memory for two CVN70 PERQs [50%]   %. STATUS: 15%
                                                   E. EFFORT: 22Days 5Hours
  3. Provide ECOs for CVN70 PERQs [20%]

  4. Establish Reliability Improvement Program [30%]

  5. Provide PERQ Maintenance Training course [0%]

  6. Provide PERQ maintenance contract for Jul-Sep [25%]
                                                          N.-Notes
  7. Provide priority purchase for VAX-780 [0%]
                                                          ↑. parent
  8. Install VAX-780 [0%]
                                                          •. top
edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 2-5: Example task frame from USS CARL VINSON

2.3.5. Other applications

For brevity, we will simply list some other applications areas we have explored:

- Training: Using ZOG nets to contain instructional material, which can also be integrated with material stored on a videodisc

- Interface for an expert system called *Airplan*, which provides decision support for the launch and recovery of aircraft on board the USS CARL VINSON

- Issue analysis: Using nets to analyze the pros and cons of decision alternatives

- Retrieval of emergency operating instructions for commercial nuclear power plants

3. Database Problems and ZOG Solutions

This section uses a set of common database problems to organize what we have learned about ZOG as a database system. Many resource-sharing problems that occur in the context of centralized systems resurface in distributed systems that are sophisticated enough to admit significant resource-sharing opportunities. One of the most important overall problems (which encompasses several of the specific problems listed below) is how users can make their work usable by others, since there are few situations in the real world where people do not depend on interaction with others to accomplish their work. This is a relatively easy problem to deal with in centralized systems; for distributed systems it becomes a major headache.

The database problems we will discuss in this section are:

- Building Problem
- Flexibility Problem
- Aggregation Problem
- Reliability Problem
- Multiple Versions Problem
- Integration Problem
- Unfamiliarity Problem
- Accessibility Problem
- Security Problem
- Conflicting Update Problem
- Change Awareness Problem

3.1. Building Problem

How do we build up the content of large, useful databases in a timely and cost-effective fashion?

Developing useful databases is a very difficult problem. ZOG addresses this problem in several ways. First, we have attempted to make the process of manually creating frames easy to learn and easy to do. Frame creation can be initiated simply by selecting an existing selection that is not currently pointing to another frame. At this point, the user can elect to create a new frame, link the selection to an existing frame, or abort the process. (New frames are created in the same subnet as the spawning parent, unless the user overrides this by specifying another subnet). If the user elects to create a new frame, he is asked what frame he wishes to copy as the initial version of the new frame. The default frame to copy is the "zero" frame in the subnet, but the user can elect to copy any frame that exists. The indexing capabilities of ZOG make it easy for users to become aware of what schematic structures exist that may be useful for copying. Once a frame is created, the user can enter the editor and make changes directly (or in the case of more specialized editors, be buffered by type checking and error checking procedures). This means, if we view a frame as a record for the moment, that users can add new fields to individual records at any time.

Second, in addition to the single frame creation mechanism, ZOG provides utilities for automatically copying and filling out schematic structures consisting of many frames, such as a large tree of frames. For example, in the on-line ZOG version of the VINSON's SORM (Ship's Organization and Regulations Manual), there exists a generic description of a task called "Getting underway". As you can imagine, putting an aircraft carrier out to sea is similar to the process of launching a spacecraft, with several days of synchronized tasks leading up to the actual departure. This task is represented in a form which can be easily read by both users and programs. Whenever the ship is to get underway, a copy of this task tree is automatically made and filled out with the actual time schedule and people involved. This instantiation of the generic form can then be used to track the status of accomplishment of the subtasks, with higher levels of the tree giving a summary of the status of whole subtrees. Ramakrishna has extensively investigated the use of schematization as an aid to developing ZOG nets [5]. He showed that *schema mechanisms* can be a more flexible tool for database construction than *type mechanisms*, since users are free to change substructure in ways that type mechanisms prohibit.

Finally, we have briefly explored the use of special programs to automatically convert existing database files, primarily document text files, into ZOG nets as another means of developing such databases. For example, a system developed by Fox and Palay [6], partially automated the construction of a library database in ZOG frame form. Their system, called BROWSE, began with an already existing bibliography database separate from ZOG, and provided a way to automatically construct ZOG frames from the bibliography entries, along with index frames which provided access to the entries according to a standard classification scheme.

3.2. Flexibility Problem

How do we allow for the representation of a wide variety of information forms, varying from highly-structured to relatively unstructured?

Most database systems are relatively specialized. As a result, the database world is polarized into databases of primarily two types: "conventional" databases, consisting of well-defined record structures; and text files, which are so unstructured that they are not often discussed in database terms. Often, text files are augmented with *ad hoc* conventions, such as special format characters, to embed the necessary structuring. For example, document text files require many meta-level symbols to augment the basic text with the structural information required by a document formatting program.

A particular problem with record-oriented databases is that a significant amount of pre-planning of the structures is required before one can begin loading data, and then massive restructuring is often required along the way. Particular problems with text files are that they are usually too large to provide efficient sharing through locking and unlocking, and that there are no built-in mechanisms for parts of files to refer to parts of other files.

ZOG allows the full generality of arbitrary text strings, but provides means for representing structure via the layout of individual text items on a frame, plus inter-frame links. Thus, ZOG represents a compromise between unstructured text files and highly structured databases. It has turned out in practice that ZOG structure is general enough to support a broad variety of functions. Any additional structure convenient for particular applications can be provided by the use of syntactic conventions. For example, field-like data is commonly embedded in text strings in a natural way, such as: "Date: 14 Feb 1984". Retrieval of these values involves parsing these strings. So far, we have been able to live with the loss of efficiency resulting from this approach to flexibility, partly because of the use of dedicated processors.

3.3. Aggregation Problem

How do we allow data to be aggregated into subsets for the purpose of performing some operation on the subset?

We have found that when data is organized in hierarchies, subtrees are by far the most frequent kind of subset that a user wants to operate on. In ZOG, one can easily perform many operations on an arbitrary subtree. simply by indicating the frame at the top of the subtree as a starting point; e.g., printing out only Section 1.2 of a document. In those cases where one wants to process a subset that is not already represented as a subtree, one can often build (either manually, or with a utility program) a new tree structure that collects together all the items of the desired subset for the sole purpose of processing them as a unit. This "constructive" approach to specifying the scope of an operation is semantically richer than many database query languages. ZOG allows the user to incrementally knit existing substructures together into complex forms, whereas the typical database query language produces list-like structures.

The *subnet* is another subsetting entity that has proved useful in ZOG. Subnets are normally defined in a functional manner; i.e., a subnet consists of all frames that jointly provide some function. These functional groupings are exactly right for many operations on subsets of frames.

3.4. Reliability Problem

How do we provide for uninterrupted access to the data in the face of failures in the hardware and software?

We have only made modest attempts to address the problem of maintaining access to information in the face of fluctuating hardware availability. Our approach has been to store additional copies of a subnet, which reside on independent machines. The user decides at the time he creates a subnet how many *secondaries* he wishes to have. Each time a frame in the primary version of a subnet is updated, the secondaries are also updated if their respective machines are functioning. When the machine with the primary version is not available, the system will automatically fall back to providing one of the secondary versions, but will not allow a secondary version to be edited. There are lots of pragmatic difficulties with a scheme as unsophisticated as ours, since we did not develop mechanisms for automatically restoring consistency among all versions as machines come back on-line.

We did develop mechanisms for backing up changed portions of the database (whole subnets) over the network, as an additional means of coping with unreliability.

3.5. Multiple Versions Problem

How do we represent multiple versions of some compound object in the database, so that we can access each version as if it alone existed, yet can still share the parts that the versions have in common?

We have found that a few simple utilities go a long way towards addressing the problem of representing and managing multiple version of an object such as a document or program. One of the utility functions made available at the bottom of each frame is the global pad "old". The "old" utility simply makes a saved copy of the current frame and attaches it to the current copy. Subsequent uses of the function continue to grow the list of saved versions in a "push-down" manner. We have established informal rules for when a copy should be saved before editing a frame; these rules are of necessity more strict for Pascal code than for prose. This "old" mechanism is relatively efficient space-wise, since copies are made only of single frames, rather than larger units as in a file-oriented system.

3.6. Integration Problem

How are multiple programs able to operate on the same data without requiring conversion or re-entry of the data in different formats? Is the data obtained from a remote machine compatible with the software on the local machine?

ZOG solves this problem by imposing a single, uniform data model on all application programs: the frame. It is best when applications are originally developed within the ZOG architecture; programs not explicitly written to work within ZOG must be converted. The preferred method is to have programs be completely frame oriented: be activated by a user making selections on frames, receive their parameters from frames, get their data from frames, and create frames as output. From the user's point of view, such programs are so homogeneous that users feel they half know how to use a program they have never used before. From a programmer's point of view, applications programs are much smaller since the task of interfacing with the user, for both input and output, is completely handled by ZOG. Conventional database systems focus more on shielding the programmer from the details of the database--interaction with the user is often not addressed.

3.7. Unfamiliarity Problem

How do users find their way in a database filled with unfamiliar data?

This is an immediate and severe problem for new users of a database; but note that in a very large database even long-time users will find large areas of the database unfamiliar.

ZOG provides an effective *browsing* capability that allows users to explore the database -- much like driving around a new town to get a feel for what the place is like. Most information in ZOG is structured in a hierarchical fashion, which means that each node in the tree acts as an index for the information below.

Browsing is valuable because:

- It provides a means of searching a database when the user has a poor model of the contents of the database.
- It works when the user doesn't really know exactly what he is looking for, but may recognize it when he sees it.
- It provides a means of solving information-processing problems that go beyond the automatic capability of the system, with the user engaging in some manual search strategy.
- It has the good side-effect of the user progressively acquiring a mental model of the

contents and structure of the database.

Browsing in ZOG is strongly distinguished from the more typical approach of providing to the user a language for making queries of the database. A good analogy comes from the sport of fishing: the user making database queries is like a rod-and-reel fisherman who baits a hook (with a query), drops the line down into the water where he can't see it, and hopes to catch something. The process involves a lot of guesswork and trial-and-error. By comparison, a user browsing through a ZOG database is like a spear-fisherman, who goes down into the water and swims around until he finds the fish he is looking for.

One of the pitfalls of browsing in a large, unfamiliar database is the possibility of getting lost. However, our experience (we have collected data on over 300,000 ZOG sessions) indicates that getting lost is not usually a severe problem with ZOG, since the user has commands available on every frame that help him get re-oriented if he ever does get lost. Also, since net-builders usually create hierarchical structures rather than arbitrary network structures, there are fewer opportunities for getting lost.

3.8. Accessibility Problem

What mechanisms do we provide for users to get access to data created by other users on the network? And given the mechanisms, how does the user locate the specific things he needs within the total collection of data?

ZOG provides accessibility through a single, distributed database structure that transparently (to the user) spans all machines on the local-area network. A single, comprehensive index structure is manually created to locate specific items in the database. This index is functionally oriented. It does not rely on knowledge of the name of the network node where the information resides, as is typically required in distributed file systems.

ZOG also provides a simple dynamic searching mechanism (via the "find" command) that searches through subsets of the database for instances of a given string, and then allows navigation among the found instances. This facility also transparently spans all machines on the local-area network.

3.9. Security Problem

How do we restrict access to sensitive information to the appropriate subset of users?

ZOG allows individual frames to be protected. Each frame has a list of owners; the creator is the initial owner, and only existing owners can add new owners (or remove themselves as an owner). An owner of a frame can mark it as write-protected, which means that only the owners will be able to modify it; or an owner can mark it as read-protected, which means that only owners can see the information at all.

An additional form of protection that ZOG provides is to restrict the execution of utility and application programs to a subset of the users. Each program can have its own list of users authorized to execute it; the list can be changed only by a special "system maintainer" user.

3.10. Conflicting Update Problem

If more than one person attempts to modify a piece of data at the same time, how do we prevent corruption of the data? How do we prevent lost efficiency of users due to being locked out from making desired changes?

ZOG provides locking of data for modification at the level of a single frame: entry to the ZOG editor reads and locks the frame, and exit from the editor updates and unlocks the frame. (The editor cannot be entered while another user already has the frame locked). Systems that provide sharing via files (which typically are much larger than ZOG frames) tend to induce an explicit "taking turns" behavior among the users, since it is so difficult to work jointly when whole files must be locked to be modified. With ZOG, it is common practice for two or three users to be simultaneously authoring a small on-line document, with very little interference.

3.11. Change Awareness Problem

How can the user be made selectively aware of recent changes made by others to the database?

When a database becomes large, recent changes can be as difficult to find as needles in a haystack. Yet it is important that a user be able to reliably find all changes made by others; otherwise, the usefulness of the database as a communications medium is seriously compromised.

The ZOG "find" mechanism can be used to create a list of all frames changed (by anyone, or by a particular person) since a given date and time. Once created, this list can be used to browse among the changed frames. Each frame carries with it the time it was last modified, and the identity of the person who made the modification. Also, there is often (by convention) a saved copy of the old version of the frame available for comparison with the new version. This change-list mechanism also has the capability to remember for each different user of the system the last time they created such a change list, so that a new list can easily be created relative to the time of the last one.

Another mechanism, called *incremental display*, can provide immediate notification of changes made by another user to the frame currently being viewed on the computer display. Incremental display works by polling the master database copy of the frame at a set interval (normally in the range of 1 to 10 seconds), watching for an increment in the frame's internal version number. When such a change is detected, the new version of the frame is compared with the old (displayed) version, and the changes are highlighted on the display.

4. Extensions to the ZOG Approach

Outside the university, we have developed a commercial system based on the ZOG approach, called the Knowledge Management System (KMS). KMS extends the ZOG approach along a number of dimensions. In this section, we cover those extensions that are most relevant to database design.

Extended model of a frame. In KMS, the model of a frame has been extended to include graphical items as well as textual items. It has a single editor which integrates text and graphics. Also, the distinction between types of textual entities (frame title, frame text, options and local pads) is no longer explicitly maintained by the system. All such distinctions are now made by visible cues, such as position or visible "tag" characters, thereby making the process of frame modification more direct.

Multiple view types. In addition to the breadth-first view that a frame provides, we are adding other view types, which are created dynamically with respect to the user's "current frame". The *linear* view is composed by starting at the current frame and traversing the subtree below it in depth-first order, gathering up a screenful of material at a time. This "page formatting" process takes an average of 3 seconds per page. Other planned view types include:

- **Table-of-Contents view**: an indented listing of the frame titles in the subtree below the current frame
- **Periscope view**: an upwards "fisheye" view to provide a view of the global context for a frame (useful if you "tunneled" to a frame via a cross-reference link)
- **Index view**: an alphabetic listing of the keywords in the subtree below the current frame
- **Return view**: a chronological listing of the frames (by title) you visited, minus frames to

which you have returned

- **Find view**: a listing of frames (by title) found by the search utility

These new views will not just be passive displays--they will be active like regular frames. Thus the user will be able to navigate through the database, invoke programs, and edit content from within these views.

Network gateways. Implementing a distributed database within a local-area network has whetted our appetite for extending the database across multiple networks. Such an architecture would permit a geographically dispersed organization to integrate all of its data into a single global database. We expect the primary issue will be how to manage information about the location of subnets in a flexible and efficient manner, but without grossly complicated mechanisms.

Imperfect but efficient locking mechanisms. Another area we are currently exploring is alternate database locking strategies for handling concurrency problems. We now suspect that in databases that are very large relative to the scope of activity of users and programs, it is acceptable to dispense with the locking of data before modification *in most cases.* What led us to question the wisdom of locking (with its unpleasant overheads) was the recognition that the probability of problems due to absence of locking was becoming small relative to the probability of other types of problems, such as hardware or operating system failures. We believe that detection and recovery mechanisms can be built to help cope with those rare cases where users make conflicting updates

Layering KMS on top of a standard database system. Periodically, we have entertained the notion of integrating KMS with a conventional database system (e.g., a relational database). If successful, this would add to ZOG the power of conventional database systems for answering well-formed queries. However, there is some doubt about whether the rapid response requirement of ZOG could be maintained if the component parts of a frame ended up being scattered all over the physical storage device.

Thinking along the lines of interfacing to existing database systems has prompted us to reexamine the appropriate role for mainframe computers in a distributed environment. We think mainframes would work well as giant ZOG "frame servers", since one of the principal advantages of mainframes (over current personal computers) is their ability to manage large quantities of inexpensive, fast secondary memory.

5. Some Recommendations

We conclude with some recommendations for database system designers based on our experience with ZOG.

- Start out with a design at some medium level of generality and generalize the system only as need dictates.
- Make things easy for humans even if it makes things hard for programs. It should be easy for humans to build large network-structured objects and to rapidly browse within such objects.
- Be willing to trade other things (e.g., efficiency of programs) for more integration of tools and users.
- Design the system to accept some (low) level of unreliability as given -- this may allow a much simpler and more efficient design. For many applications, maintaining perfect integrity and consistency of a database may not be worth the considerable costs to do so.
- Use powerful hardware to avoid complex software designs and costly software optimization.

6. Acknowledgements

We wish to acknowledge the contributions of many people over the years. Those who have been involved with ZOG at CMU: Allen Newell, George Robertson, Kamila Robertson, Elise Yoder, Peter Lieu, Sandy Esch, Patty Nazarek, Angela Gugliotta, Marilyn Mantei, Kamesh Ramakrishna, Roy Taylor, Mark Fox, and Andy Falay. Those officers from the USS CARL VINSON who worked with us at CMU: Mark Frost, Paul Fischbeck, Hal Powell, Russ Shoop, and Rich Anderson. Captain Richard Martin, Captain Thomas Mercer, Cdr Ted Kral, Lt Brian Mackay and other officers and crew of the USS CARL VINSON. And finally, Marvin Denicoff from the Office of Naval Research, our original ZOG research sponsor. We would also like to thank Elise Yoder and Mark Frost for their extensive comments on this paper.

This work was supported by the Office of Naval Research under contract N00014-76-$\overset{C}{\wedge}$0874. It was also partially supported by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the Defense Advanced Research Projects Agency, or the U.S. Government.

7. References

[1] G. Robertson, D. McCracken, and A. Newell, "The ZOG approach to man-machine communication," *International Journal of Man-Machine Studies*, vol. 14, pp. 461-488, 1981.

[2] D. McCracken and R. Akscyn, "Experience with the ZOG human-computer interface system," to appear in *International Journal of Man-Machine Studies*, July 1984.

[3] A. Newell, D. McCracken, G. Robertson, and R. Akscyn, "ZOG and the USS CARL VINSON," *Computer Science Research Review*, Computer Science Department, Carnegie-Mellon University, pp. 95-118, 1981.

[4] B. Shneiderman, "Direct manipulation: a step beyond programming languages," *IEEE Computer*, pp. 57-69, 1983.

[5] K. Ramakrishna, *Schematization as an Aid to Organizing ZOG Information Nets*, Ph.D. thesis, Computer Science Department, Carnegie-Mellon University, 1981.

[6] M. Fox and A. Palay, "The BROWSE system: an introduction," *Proceedings of the Annual Conference of the American Society of Information Science*, Minneapolis, pp. 183-193, October 1979.