Experiments with OVAL: A Radically Tailorable Tool for Cooperative Work

Thomas W. Malone, Kum-Yew Lai, and Christopher Fry

CCS TR #132, Sloan School WP # 3462-92

August, 1992

Acknowledgments
This research was supported by Digital Equipment Corporation, the National Science Foundation (Grant No. IRI-8903034), Matshushita Electric Industrial Co., Boeing, Information Resources, Inc., Electronic Data Systems, Apple Computer Company, and the corporate members of the MIT International Financial Services Research Center.

We are indebted to Terry Winograd for suggesting the acronym "Oval" after hearing about the four components of our system and to John McDermott for suggesting the term "radically tailorable" as a description of the Oval system. The Oval system has benefited from the suggestions and work of numerous people over many years, including most recently: Mark Ackerman, Geoffrey Bock, Kevin Crowston, Troy Jordan, Jintai Lee, Charley Osborne, Dave Park, Mike Plusch, and Paul Resnick.

Experiments with Oval: A Radically Tailorable Tool for Cooperative Work

Thomas W. Malone, Kum-Yew Lai*, and Christopher Fry

MIT Center for Coordination Science

Appeared in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '92)*, Toronto, Ontario, Canada, November 1992.

Experiments with Oval: A Radically Tailorable Tool for Cooperative Work

Thomas W. Malone, Kum-Yew Lai*, and Christopher Fry

ABSTRACT

This paper describes a series of tests of the generality of a "radically tailorable" tool for cooperative work. Users of this system can create applications by combining and modifying four kinds of building blocks: *objects, views, agents*, and *links*. We found that user-level tailoring of these primitives can provide most of the functionality found in well-known cooperative work systems such as gIBIS, Coordinator, Lotus Notes, and Information Lens. These primitives, therefore, appear to provide an elementary "tailoring language" out of which a wide variety of integrated information management and collaboration applications can be constructed by end users.

INTRODUCTION

For as long as computers have been used for practical tasks, software designers have tried to match their systems to the situations in which they are used. This problem is perhaps nowhere more important than in designing software to support groups of people working together [8]. In some approaches to this problem, software designers study organizations using systems analysis techniques (e.g., [23]) or social science methodologies (e.g., [5], [20], [18]). In participatory design approaches to this problem, the eventual users of a system (who already know about the context of its use) work with software developers in its initial design (e.g., [6], [19]).

An even more extreme version of the participatory design approach is not just to *involve* users in design but to let them *become* designers by giving them end-user programming tools (e.g., [13], [4], [17], [12]). For instance, with products like spreadsheets and Hypercard, end-users, who have no specific training in software development, can

* Current address: McKinsey & Co., 18/F Two Exchange Square, Central, Hong Kong.

create for themselves more and more of the kinds of applications that would previously have required substantial work by professional programmers.

In this paper, we focus on one important class of such end-user programming tools. We call this class of systems *radically tailorable* since they allow end users to create a wide range of different applications by progressively modifying a working system. Radically tailorable systems differ from "ordinary" tailorable systems (such as word processing programs with "Preferences" parameters) in the degree to which users can create a wide range of substantially different applications. For instance, starting with the same blank spreadsheet, users can create applications ranging from personal budgeting to sales forecasting to corporate finance.

Radically tailorable systems also differ from conventional programming languages (including typical fourth generation languages) in that end users progressively modify a working version of an application (such as a blank spreadsheet), instead of specifying instructions in some programming language. In this way, radically tailorable systems reduce the "cognitive distance" between using an application and designing it (c.f., [7]).

Radically tailorable systems are not the only approach to matching cooperative work tools to their contexts of use, but we believe they will be an increasingly important one. One of the key problems in designing radically tailorable tools is picking a set of building blocks at the "right" level of abstraction. That is, the building blocks should not be so low-level that they require significant effort to do anything useful, nor so high-level that they require significant modification whenever the users' needs change (see [3]).

In this paper, we describe a set of experiments with one such set of building blocks. The building blocks are embodied in a radically tailorable system for cooperative work called Oval, the name of which is an acronym for the four key building blocks of the system: *objects*, *views*, *agents*, and *links*.[1] The experiments we describe are tests of the generality of these building blocks: What is the range of different kinds of applications that can be implemented "naturally" with these building blocks and the user level tailoring facilities provided by the Oval system?

To do this, we first constructed a wide range of relatively simple applications for tasks such as project management, software bug tracking, meeting scheduling, and personal information management. These simple applications are

1 Oval is based upon the Object Lens system ([9], [14]). However, we picked a new name to reduce the widespread confusion between Information Lens and Object Lens. Information Lens [16] was a system for intelligent mail sorting. Oval, as this paper describes, is a much more general system in which Information Lens is only one possible application.

not described here. Instead, we focus in this paper on four applications that are already believed by many people to be valuable for supporting cooperative work. Three of these applications were developed elsewhere: gIBIS [2], Coordinator [21], and Notes [11]. One was developed in our laboratory: Information Lens [16].

For each of these systems, we attempted to emulate as much as possible of the functionality of the original system using only the user level facilities provided by Oval. We also identified (and in some cases performed) system level modifications necessary to emulate the functionality of the original systems more closely. With a few exceptions, the basic functionality of the original systems was achieved with only user level modifications.

These tests demonstrate, therefore, that the four key components of Oval (objects, views, agents, and links) provide a *tailoring language* with which a wide variety of cooperative work tools can be constructed. It would, of course, be no surprise to say that we could implement all these applications in a general purpose programming language or that primitives like objects, views, agents, and links were helpful in doing so. The surprising thing, we believe, is that all these applications can be implemented using only the extremely restricted and simplified tailoring language provided by Oval.

OVERVIEW OF OVAL

Oval is based upon four key building blocks (see [9] and [14] for much more detailed descriptions):

(1) Semistructured *objects* represent things in the world such as people, tasks, messages, and meetings. Each object includes a collection of fields and field values and a set of actions that can be performed upon it. The object types are arranged in a hierarchy of increasingly specialized types with each object type inheriting fields, actions, and other properties from its parents in the hierarchy. The objects are semistructured in the sense that users can fill in as much or as little information in different fields as they desire and the information in a field is not necessarily of any specific type (e.g., it may be free text, a link to another object, or a combination of text and links). Users see and manipulate these objects via a particularly natural form of template-based interfaces.

(2) User customizable *views* summarize collections of objects and allow users to edit individual objects. For instance, users can select the fields to be shown in a table display of a collection of objects, or they can select the links to be used to create a network display of the relationships between objects. A calendar display can be used to summarize objects with dates in one of their fields. Any appropriate display format (e.g., table, network, or calendar) can be used to show any collection of objects in any field of any object. Typically, these display formats are used for the "Contents" field of "Folder" objects.

(3) Rule-based *agents* perform active tasks for people without requiring the direct attention of their users. Agents can be triggered by events such as the arrival of new mail, the appearance of a new object in a folder, or the arrival of a pre-specified time. When an agent is triggered it applies a set of rules to a collection of objects. Rules contain descriptions of the objects to which they apply and actions to be performed on those objects. Actions include general actions such as moving, mailing, and deleting objects or object-specific actions such as loading files or responding to messages.

(4) *Links* represent relationships between objects. For example, users can use links to represent relationships between a message and its replies, between people and their supervisors, and between different parts of a complex product. Users can follow these hypertext links by clicking on them, and the knowledge represented by the links can be used by rules or in creating displays.

User tailoring

The primary user level modifications to the system include: (1) defining new object types, (2) adding fields to existing object types, (3) selecting views for objects and collections of objects (from a prespecified set of display formats), (4) specifying parameters for a given view (such as which fields to show), (5) creating new agents and rules, and (6) inserting new links.

Sharing information

There are three primary ways for people to save and share information in Oval: (1) They can save any collection of objects in a file which they (or other people) can load later. For instance, all the objects linked (directly or indirectly) to a given object can be automatically collected and saved in a file. (2) They can mail any collection of objects back and forth to each other in messages. When users load a previously saved file or receive a message containing objects, the identity of the objects is preserved (i.e., pre-existing links will point to the newest versions of the objects and the previous versions will be stored as "previous versions"[2]). (3) We have also implemented a rudimentary version of "live" sharing of objects stored on remote databases. Even without this capability, agents can be used to automatically mail and sort "shared" objects thus providing many of the benefits of live sharing.

Implementation status

Oval is implemented in Macintosh Common Lisp on networked Apple Macintoshes. As of this writing, various versions of this system have been used intermittently by up to six people in our research group over a period of approximately 2 years, and by numerous other people for shorter periods. Over 100 copies of the software have been distributed to other researchers and developers for demonstration purposes.[3]

TEST APPLICATIONS

To test our hypothesis that Oval is radically tailorable, we used the system to try to implement the functionality of a variety of cooperative work applications for which descriptions have been previously published. In this section, we

[2] Users are notified of these changes, and if they desire, can undo them in specific cases.

[3] For information on how to obtain a copy of the software for research purposes at no charge, contact Peter Richards, MIT Technology Licensing Office, E32-300, 28 Carleton Street, Cambridge, MA 02139 (Telephone: (617) 253-6966. Email: par@eagle.mit.edu.)

will describe briefly how we implemented the major features of each application and what features of the original applications we did not implement. The first application is described in somewhat more detail than the others to illustrate how the system works. More detailed examples of how new applications are created can be found in [9] and [14]. Note that we would not expect beginning users of a system like Oval to be immediately able do all the tailoring needed for some of these applications. Some of the applications require the use of tailoring features (such as defining new object types) that we would expect of experienced users.

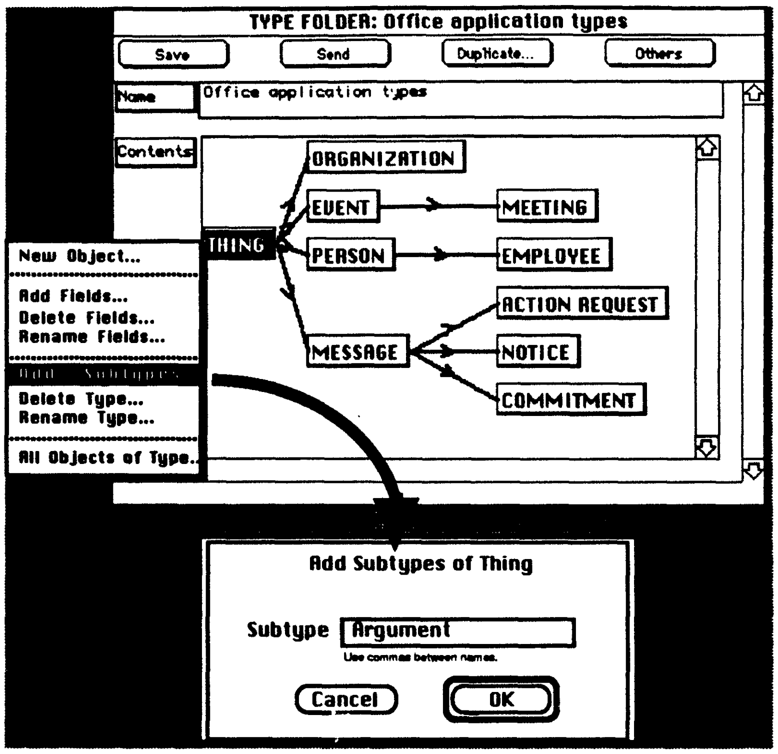
For each application, we were primarily concerned with whether the overall user interface paradigm provided by Oval could accommodate in a "natural" way the primary functionality of the application. We did not attempt to mimic exact details of screen layout and command names. Similarly, since our system is only a research prototype, we did not attempt to replicate the level of attention to robustness, speed, access controls, and so forth present in the commercial products we analyzed. In all cases, however, we tried to be faithful to the spirit of the original systems.

gIBIS – Argumentation Support

gIBIS [2] is a tool for helping a group explore and capture the qualitative factors that go into making decisions. Elements of a policy analysis in gIBIS are represented as a network containing three types of nodes: Issues, Positions, and Arguments. Each Issue may have several Positions that "Respond to" it, and each Position, in turn, may have various Arguments that "Support" or "Object to" it.

Defining new object types and creating examples of them

To emulate gIBIS in Oval, we first defined the three types of objects used by gIBIS: Issues, Positions, and Arguments. For instance, to define the new type of object called "Argument", we performed the actions illustrated in Figure 1. First, we selected the basic object type called "Thing" and then chose the "Create Subtype" action (Figure 1(a)). To create individual examples of this type, we selected the new type and chose the "New object" action. The new Arguments have the fields "Name", "Keywords", and "Text" by default, since these fields are present in all Things. To add the fields (like "Supports", "Objects to", and "Entered by") that are present in Argument objects but not in all Things, we used the "Add Field" action on one of the new Argument objects (Figure 1(b)). Finally, we filled in the fields of this (and other objects) by typing and by adding links (Figure 1(c)).


TYPE FOLDER: Office application types

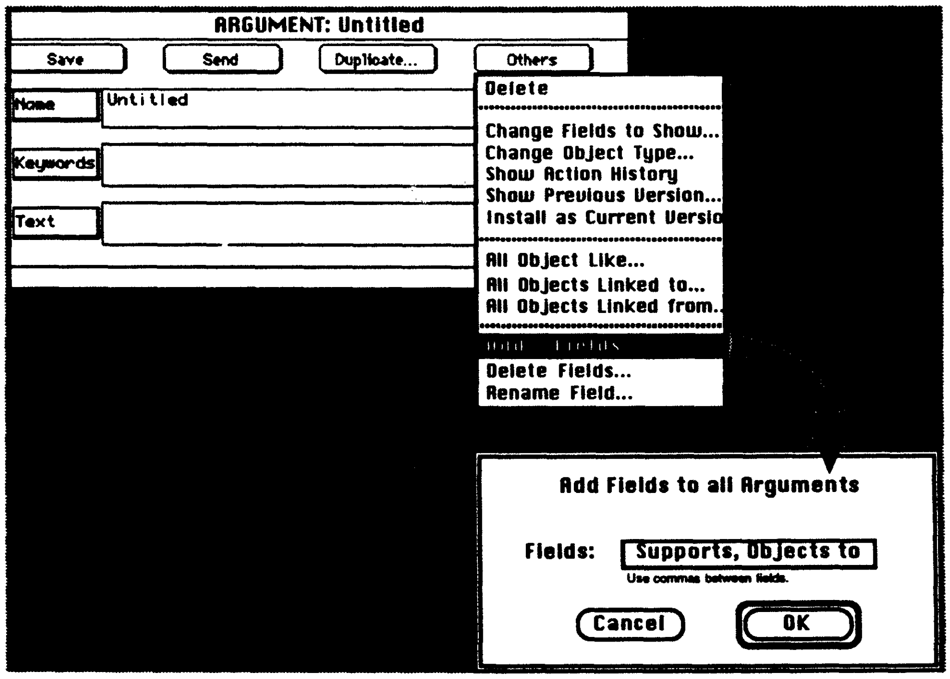
Name: Office application types

Add Subtypes of Thing

Subtype: Argument

Use commas between names.

Figure 1(a). To define a new type of object, users create it as a subtype of some existing object type. (Note: In this and subsequent figures, the heavy curved arrow is added for clarity. It is not part of the actual screen display.)


ARGUMENT: Untitled

Name: Untitled

Keywords:

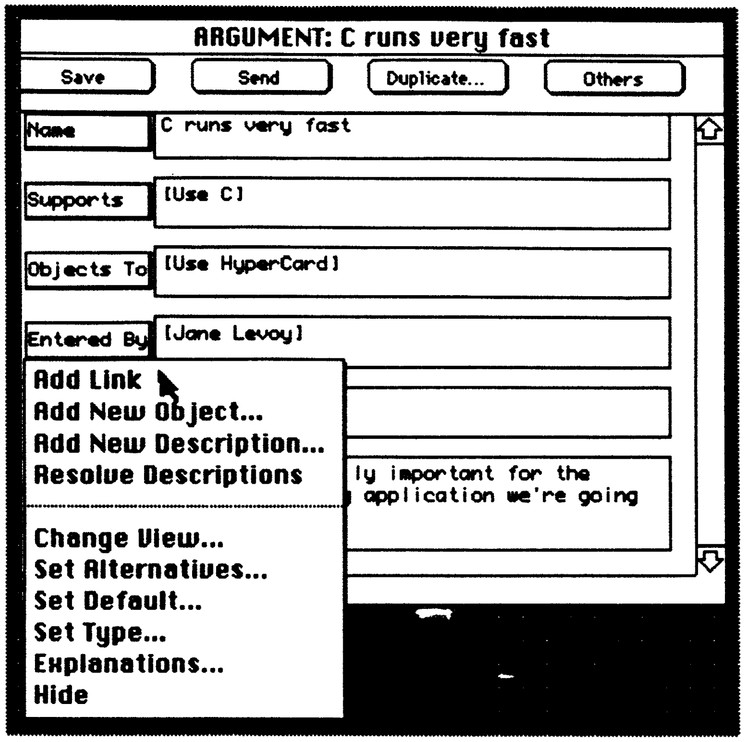
Text:

Add Fields to all Arguments

Fields: Supports, Objects to

Use commas between fields.

Figure 1(b). Users can add fields to an object type with the "Add Fields" action on any instance of that type. The fields then appear in all objects of that type.


ARGUMENT: C runs very fast

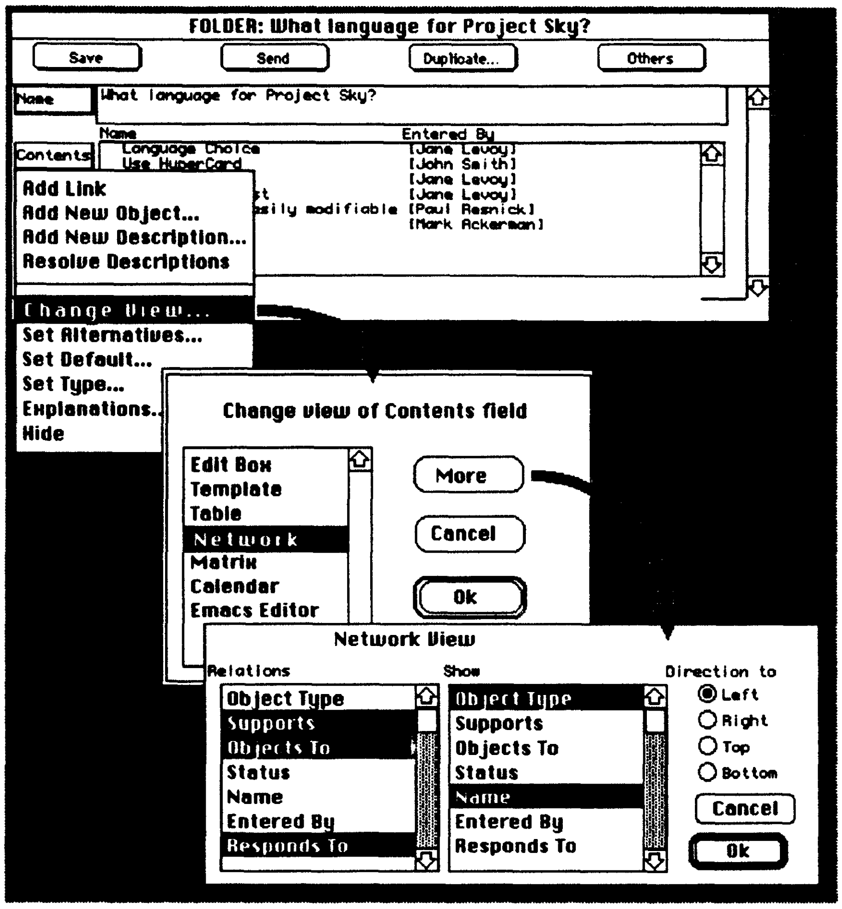
Name	C runs very fast
Supports	[Use C]
Objects To	[Use HyperCard]
Entered By	[Jane Levoy]
	ly important for the application we're going

Figure 1(c). Users can fill in the fields of an object by typing or by inserting links to other objects.

The square brackets in Figure 1(c) indicate "live" links to other objects that can be traversed by clicking on them. To add these links in the first place, we simply selected the "Add link" action in a field and then pointed to the object to which the link goes.

Viewing collections of objects

In the original gIBIS system, users can graphically see the relationships between the nodes in an argument network. To do this with Oval, we selected the "Change View" action for a field containing the nodes and then chose the "Network" display format (see Figures 2 and 3). This format allows us to choose the fields from which links will be shown in the network. For instance, in Figure 2, we chose to show links from three fields,"Supports", "Objects to" and "Responds to." We also chose to display in each node the "Name" and "Object Type" fields. The result is shown in Figure 3.



Name: What language for Project Sky?

Contents:

Name	Entered By
Language Choice	[Jane Levoy]
Use HyperCard	[John Smith]
	[Jane Levoy]
t	[Jane Levoy]
sily modifiable	[Paul Resnick]
	[Mark Ackerman]

Network View — Relations: Object Type, Supports, Objects To, Responds To; Show: Object Type, Name; Direction to: [✓ Left] Right Top Bottom

Figure 2. Users can "Change View" on any field. Here, the "Network" view was selected. Then, two other choices were made: (1) the fields from which links will be used to construct the network and (2) the fields to be shown in the nodes of the network

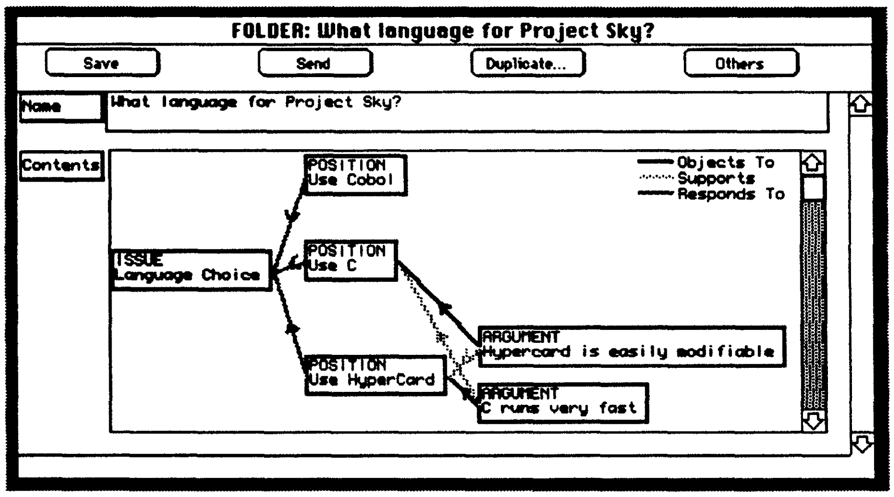


Figure 3. This network view, the result of the choices made in Figure 2, shows the relationships between Issues, Positions, and Arguments as in the gIBIS system.

Note that this simple method of tailoring a network view can be used for any collection of objects whose relationships to each other are represented by links (e.g., organization charts, PERT charts, and software module calling relationships). In one sense, of course, there is nothing new about this notion of using general display formats for many kinds of objects. We have been genuinely surprised, however, at how widely useful and powerful this feature is when users can apply it themselves to create new applications.

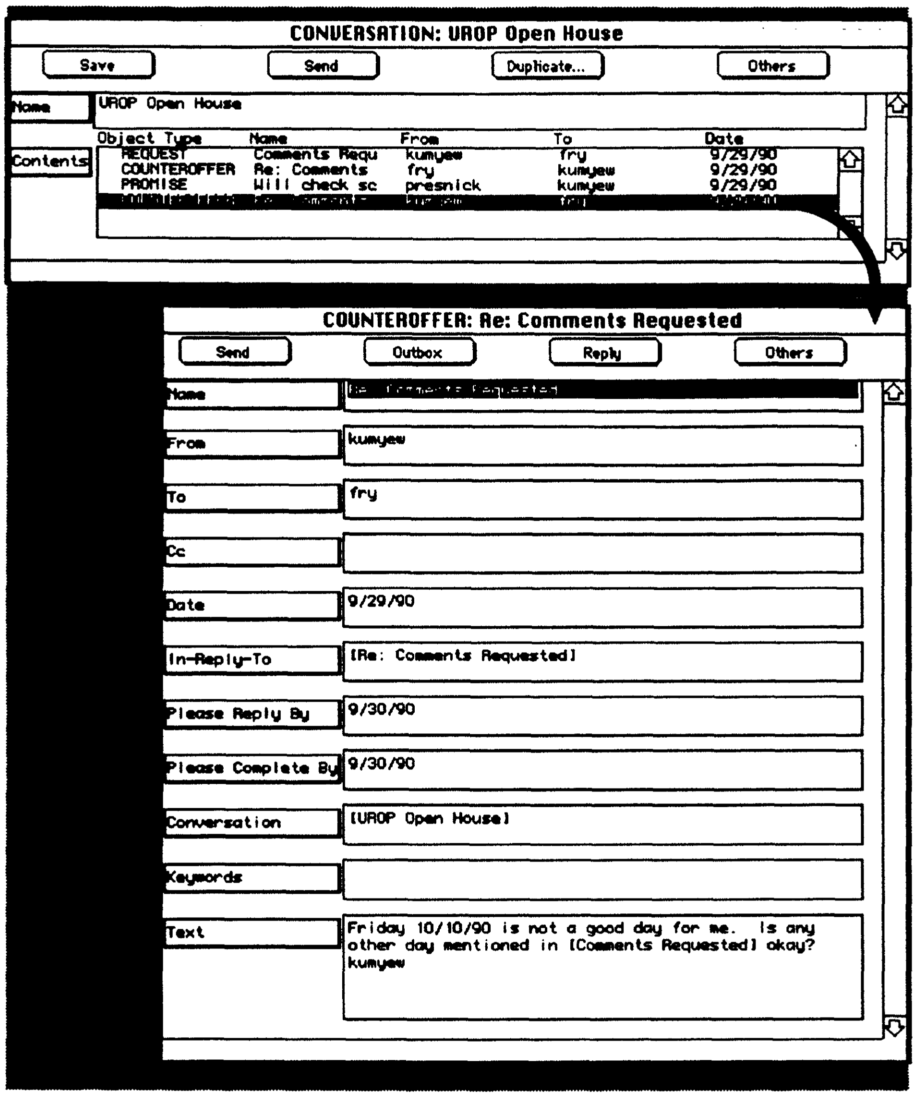
Features of the original system not included in our application

We omitted one feature of the original gIBIS system for aesthetic reasons: The original system included a distinction between primary and secondary links of each type. We could have implemented this simply by defining additional fields (e.g., "Supports (primary)" and "Supports (secondary)"), but we chose not to, in order to reduce the complexity of the system.

In addition to a shared "live" database, gIBIS has two other features which would have required new system level programming to implement: (1) aggregate nodes (the ability to automatically collapse a group of nodes into one aggregate node), and (2) a node index that shows nodes in outline format (with indentations indicating their relations). These features did not seem essential to the main point of the application, and we did not include them in our implementation. However, they would both have been quite consistent with the overall Oval paradigm. Aggregate nodes could be implemented with two new actions on folders (one to create aggregates and one to break them apart). An outline display format would be reusable in many other applications as well and would, therefore, be a useful addition to the current display formats for collections of objects (tables, networks, calendars, and matrices).

Coordinator – Conversation structuring and task tracking

The Coordinator is an electronic mail-based system that helps people structure conversations and track tasks ([22], [1]). To emulate the functionality of the Coordinator in Oval, we first made three system level modifications to automatically group a series of messages into "Conversation" folders (see Figure 4): (1) When users create a new message (without it being a reply to an old one), a new conversation folder is automatically created. This folder contains the new message, and the conversation field of the new message is also linked to this folder. (2) When users reply to a message of a given type, they are presented with a choice of message types for their reply. (The choices to be presented are specified in a user-modifiable field called "Reply Types", which is not shown in the view used in Figure 4.) (3) After users select a reply type, the Conversation field in the reply message is automatically linked to the Conversation folder specified in the original message, and the reply is inserted in the Conversation folder.



CONVERSATION: UROP Open House
Save
Send
Duplicate...
Others

Name	UROP Open House

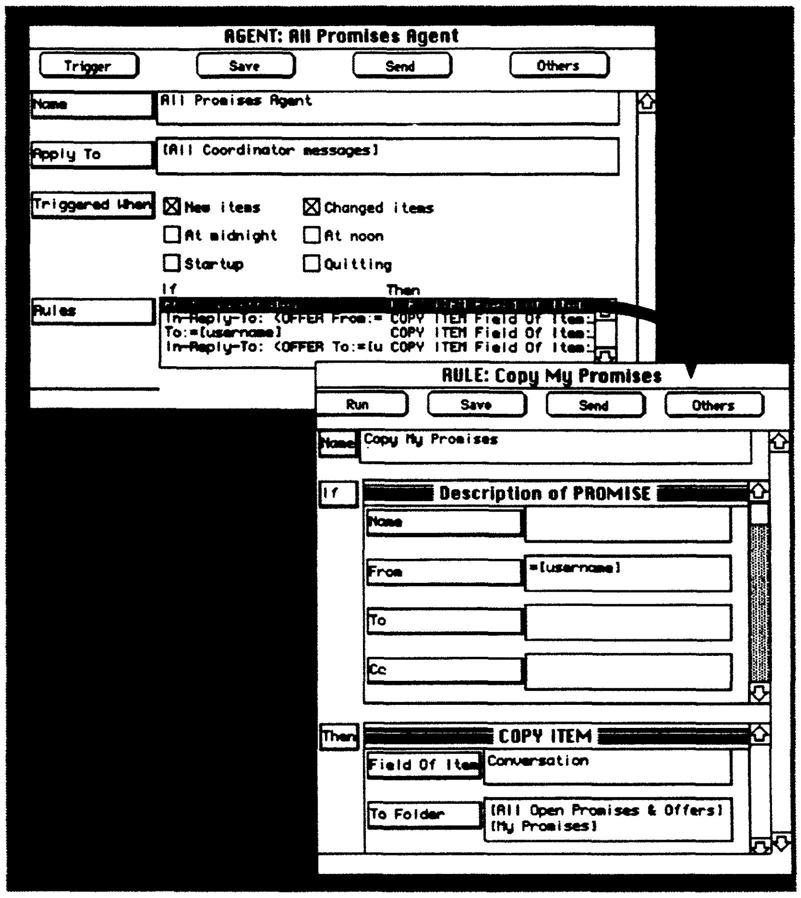
Object Type	Name	From	To	Date
REQUEST	Comments Requ	kumyew	fry	9/29/90
COUNTEROFFER	Re: Comments	fry	kumyew	9/29/90
PROMISE	Will check sc	presnick	kumyew	9/29/90
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

COUNTEROFFER: Re: Comments Requested
Send
Outbox
Reply
Others

Name	Re: Comments Requested
From	kumyew
To	fry
Cc	
Date	9/29/90
In-Reply-To	[Re: Comments Requested]
Please Reply By	9/30/90
Please Complete By	9/30/90
Conversation	[UROP Open House]
Keywords	
Text	Friday 10/10/90 is not a good day for me. Is any other day mentioned in [Comments Requested] okay? kumyew

Figure 4: A conversation folder contains messages within a conversation, like those in the Coordinator.

After these system-level changes to add conversations (which can be useful in any messaging application), the other main functionality of the Coordinator can be added by user-level tailoring. For instance, we added 15 new message types and defined their reply types to provide the same conversational sequences (such as Conversations for Action) used in the Coordinator. The Request message type, for example, has reply types like Counteroffer, Promise, and Decline.



Name: All Promises Agent

Apply To: [All Coordinator messages]

Triggered When: [✓ New items] [✓ Changed items] ☐ At midnight ☐ At noon ☐ Startup ☐ Quitting

Rules:

If	Then
[illegible]	[illegible]
In-Reply-To: <OFFER From:=	COPY ITEM Field Of Item:
To:=[username]	COPY ITEM Field Of Item:
In-Reply-To: <OFFER To:=[u	COPY ITEM Field Of Item:

Name: Copy My Promises

If: Description of PROMISE

Name	
From	=[username]
To	
Cc	

Then: COPY ITEM

Field Of Item	Conversation
To Folder	[All Open Promises & Offers] [My Promises]

Figure 5: An agent that tracks conversations involving promises. The rule shown puts conversations that include Promise messages from the user into the appropriate folders.

To provide the various kinds of summary displays in the Coordinator we created 23 folders (such as "Open Matters" and "My promises"), and 14 agents with 47 rules that move messages into and out of these folders. For instance, Figure 5 shows an agent that moves conversations into the folders for various kinds of promises. To create an agent like this one, users fill in the fields shown in Figure 5: a folder to which the agent applies, one or more triggering conditions, and a set of rules. Then, when the agent is triggered, it will apply the rules to the objects in the "Apply to" folder. Rules, like agents, are created with a straightforward sequence of menu-picks and form filling. The rule shown in Figure 5, for instance, moves conversations with promises from the user into two folders: "My Promises" and "All Open Promises and Offers".

Some of the agents make use of "all-objects-of-a-type" folders in Oval. An all-objects-of-a-type folder is one which is maintained by the system so that it always contains all objects of a specified type. For example, the "All Coordinator messages" folder is dynamically maintained so that it always contains all the Coordinator messages in Oval.

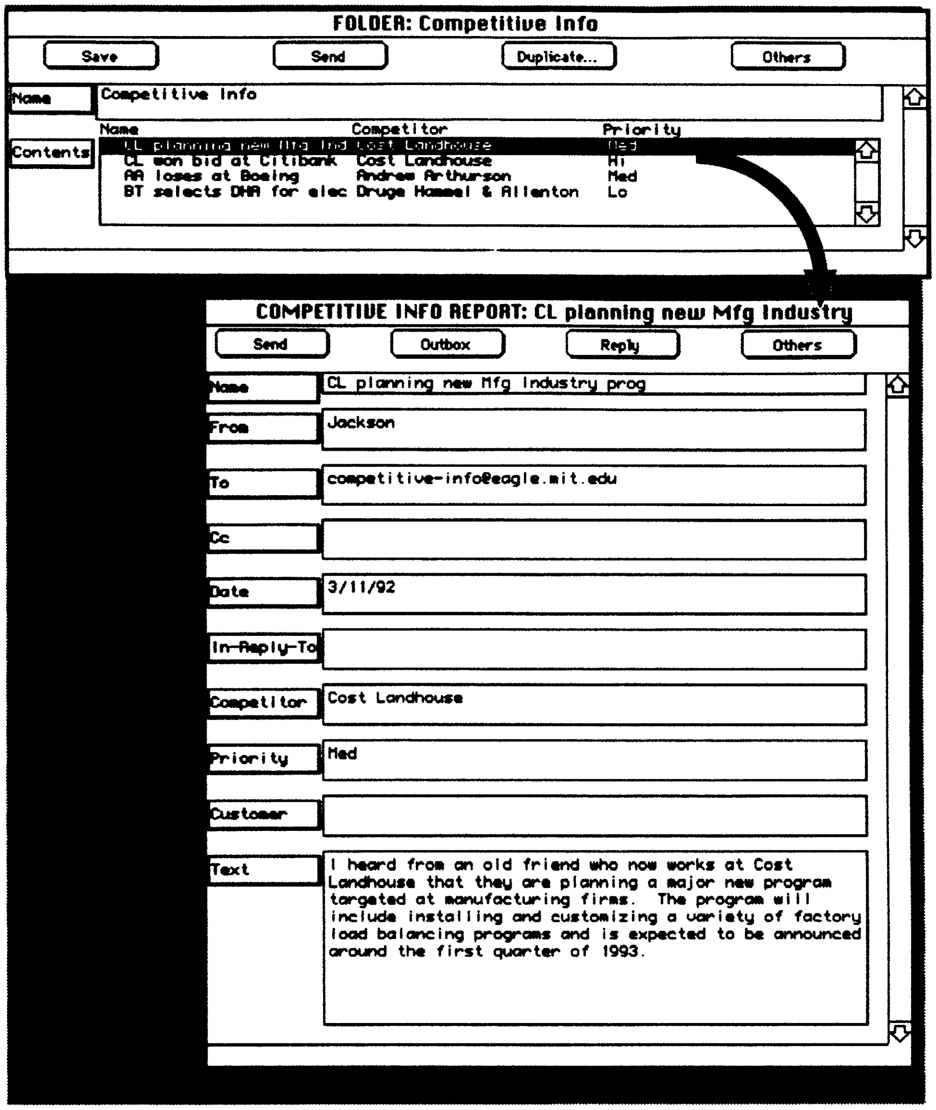
Features of the original system not included in our application

Unlike the original Coordinator, our application does not provide explicit support for delegation. Instead, someone who has received a request can delegate it to someone else "manually" by sending a new request that contains a link to the original one.

There are also several detailed features of the Coordinator not included in our application. For instance, unlike the Coordinator, our application does not enforce the restriction that people who are only copied (cc:'ed) in a conversation cannot change its state. Furthermore, there are some cases where users of our application would need to do minor things "manually" that are done automatically in the Coordinator. For instance, if someone wants to cancel a request before having received a reply to it, they would have to manually address the cancel message and insert it in the original conversation.

Notes – Semistructured information sharing

Of the other systems analyzed here, Lotus Notes [11] is the most similar to Oval. It is also similar to earlier computer conferencing systems with the important additions that (1) the documents in a database are semistructured templates (with optional "hot links" to other documents) and (2) the documents in a database can be filtered and summarized according to various user-definable views. Like Oval, Notes can be tailored for many different applications. The templates in Notes are equivalent to object types in Oval; the databases in Notes are equivalent to folders in Oval; and the views in Notes are equivalent to views of collections of objects in Oval. Notes views are all variations of a kind of "outline" display format that does not currently exist in Oval, but is similar to the table format display that does exist (see Figure 6). Notes does not include any of the other folder display formats in Oval such as networks and calendars. Similarly, even though Notes allows very knowledgeable users to do certain kinds of sorting and filtering using views, it does not appear to be designed to make this easy for end users, and it does not have active agents like those in Oval.



Name	Competitive Info

Name	Competitor	Priority
CL planning new Mfg Ind	Cost Landhouse	Med
CL won bid at Citibank	Cost Landhouse	Hi
AA loses at Boeing	Andrew Arthurson	Med
BT selects DHA for elec	Druge Hammel & Allenton	Lo

Name	CL planning new Mfg Industry prog
From	Jackson
To	competitive-info@eagle.mit.edu
Cc	
Date	3/11/92
In-Reply-To	
Competitor	Cost Landhouse
Priority	Med
Customer	
Text	I heard from an old friend who now works at Cost Landhouse that they are planning a major new program targeted at manufacturing firms. The program will include installing and customizing a variety of factory load balancing programs and is expected to be announced around the first quarter of 1993.

Figure 6. A table format display summarizing documents like those in a Notes database.

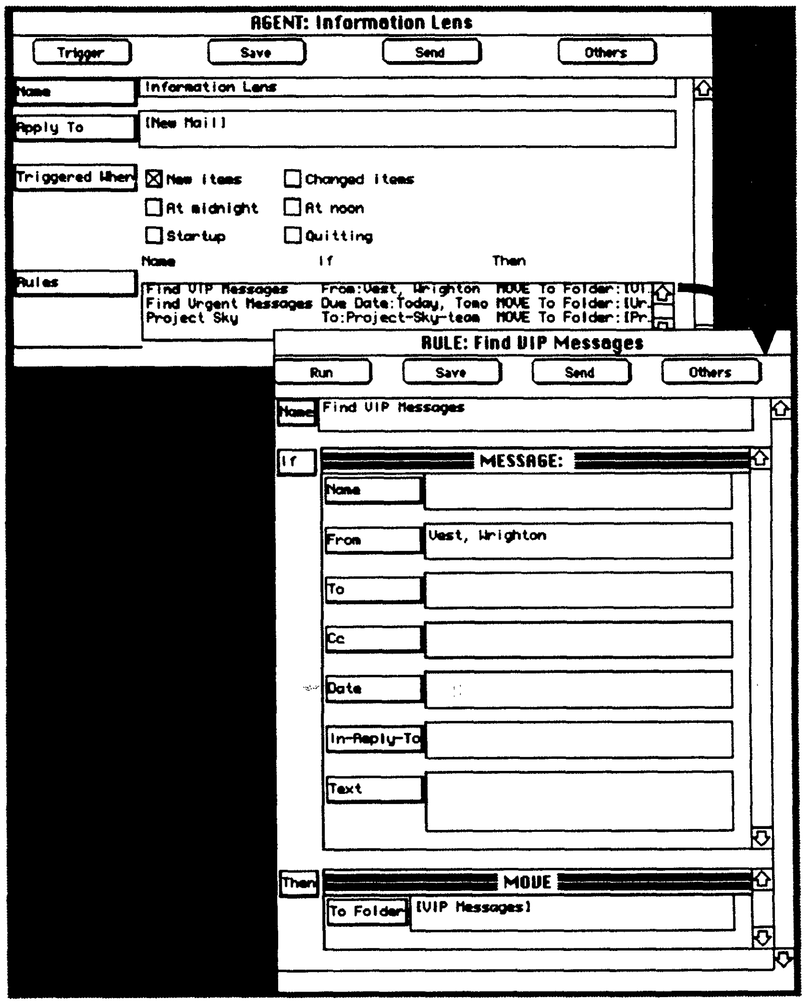
Unlike Oval, Notes is based on a database replication algorithm which approximates "live sharing" of documents. Also, as a commercial product, Notes includes a number of features such as access controls, "rich text" fields, and calculated fields. Even though these features are not currently part of Oval, they would be consistent with its overall user interface paradigm and useful for many other applications.

Information Lens -- Intelligent mail sorting

Information Lens ([16], [15]) helps users by automatically filtering and sorting incoming electronic messages. It also helps users create messages using a set of (optional) semistructured message templates. Implementing

Information Lens in Oval is quite straightforward. For instance, users can define new message types as subtypes of Message in the object type hierarchy and create rules to filter and sort messages (see Figure 7).

In addition to the local mail sorting agents, the original Information Lens included a second kind of agent that ran on servers and selected potentially interesting messages for a given user from a stream of messages addressed to "Anyone." Implementing remote agents like these would require some additional systems level programming which we have not yet done in Oval but which would be consistent with the overall framework of Oval and useful in other applications.



Name: Information Lens
Apply To: [New Mail]
Triggered When: [✓ New items] Changed items; At midnight; At noon; Startup; Quitting

Rules:

Name	If	Then
Find VIP Messages	From:Vest, Wrighton	MOVE To Folder:[VI.
Find Urgent Messages	Due Date:Today, Tomo	MOVE To Folder:[Ur.
Project Sky	To:Project-Sky-team	MOVE To Folder:[Pr.

Name: Find VIP Messages
If: MESSAGE:
Name:
From: Vest, Wrighton
To:
Cc:
Date:
In-Reply-To:
Text:
Then: MOVE
To Folder: [VIP Messages]

Figure 7. Examples of an agent and a rule like those in Information Lens.

Summary

Table 1 summarizes the results of the four application experiments described above. In three of the cases (gIBIS, Notes, and Information Lens), some of the features of the previous systems were not included in our emulations. It appears, however, that the functionality omitted in these cases was not essential to the main point of the applications. Also, in all three cases, the functionality omitted would fit naturally within the user interface paradigm provided by Oval and would be reusable in other applications. In the other case (Coordinator), we performed modest amounts of system level programming and believe we emulated all of the major functionality of the original system. The new system level functionality implemented in this case (Conversation folders) should also be generally useful in other Oval applications. Overall, therefore, the primary functionality of the previous systems seems to be captured reasonably well by user-level modifications within the user interface paradigm provided by Oval.

DISCUSSION

Limitations of experiments

Before proceeding, it is important to note several limitations of our experiments. First, even though the experiments establish that the range of applicability of systems like Oval is large, they do not allow us to characterize its limits. In fact, we suspect that it is difficult, in principle, to characterize the limits of radically tailorable tools since ingenious users will always be able to think of unexpected new uses for them.

Second, we have not formally studied the *usefulness* of the applications that could be created with systems like Oval. Instead, since we emulated other systems that are w:dely believed to be useful, we assume that systems like Oval would be at least as useful as these previous systems are individually. We do not yet know how much additional benefit will come from combining these and other applications that can be created with the system.

	User-level modifications				**System-level modifications**
System	*Object Types*	*Folders*	*Agents*	*Rules*	
gIBIS	3	1 per discussion	1	1 per query	Aggregate nodes**;"Outline" folder display format**; Live sharing †
Coordinator	19	23	14	47	Conversation folders; Reply types
Notes	*	*	*	*	"Outline" folder display format**; Live sharing †
Information Lens	*	*	1	*	"Anyone" agents**

Table 1. A summary of the modifications required to emulate the systems described in each experiment.
(* = Application specific. ** = "Optional" feature; not implemented. † = Implementation in progress)

Finally, and perhaps most importantly, we have not formally tested the *usability* of this system. That is, we tested whether the system made it *possible* for knowledgeable users to implement the applications using only the user level tailoring facilities. As the descriptions above indicate, the user level tailoring facilities have "face validity" as being simple and easy to use. Another important kind of test, however, would be to examine how well typical users at various levels of sophistication could actually create such applications for themselves.

In practice, we expect that radically tailorable systems like this will be used in very different ways by end users, by power users, and by programmers ([17], [12]). Since the system provides a wide spectrum of tailoring options, we believe that many users would make only minimal changes to applications developed by others, while some power users would develop applications for other people, and programmers would use the system to dramatically reduce the time and effort required to develop completely new applications. In fact, even if the Oval system were not tailorable at all by typical end users, radically tailorable systems like this would still be very useful for programmers as rapid prototyping and iterative development tools.

Lessons from experiments

What can we learn from these experiments? We believe that two of the most important lessons have to do with the power of the four primitives with which we started:

(1) Objects, views, agents, and links provide a kind of elementary "tailoring language" for user interfaces to information management and cooperative work applications. All of the previous systems we analyzed had some form of semistructured objects and some form of summary displays analogous to our views. Most of the systems also had some of the functionality provided by our agents and links. With the exception of Notes, however, the systems were built to handle only specialized kinds of objects (e.g., messages or argumentation nodes) and certain display formats (e.g., tables). By generalizing to the abstract level of semistructured objects, customizable views, rule-based agents, and links, we achieve a great increase in scope of applications with only a relatively modest increase in complexity. For instance, by implementing views (such as tables, networks, matrices, and calendars) once in a way that end users can then apply to any kind of information, a vast amount of programming effort is avoided and the end users' power is significantly increased. We were genuinely surprised at the power of this approach, and we believe that the generality of these basic primitives and their combination methods is not yet widely appreciated.

(2) Radical tailorability facilitates integration. In addition to the versatility that a radically tailorable system provides, there is another important benefit of constructing a variety of different applications within the same system: It is much easier to integrate the different applications. For instance, discussions facilitated by systems like

gIBIS often lead to action items, the status of which a group might want to track with systems like Notes or the Coordinator. In an integrated system like Oval, it is a straightforward matter to link these action items to the discussions that generated them and vice versa.

CONCLUSION

Even though the primitives of objects, views, agents, and links provide a powerful basis for constructing information management and cooperative work applications, there are many difficult problems to be solved before the full promise of this approach is realized. For instance, numerous difficult issues arise in trying to manage large numbers of linked objects in a shared, distributed network. One particularly important aspect of this problem involves how to manage the evolution of partially shared object type definitions (e.g., [10]). Finally, even if the approach described here were successful in completely eliminating the programming effort needed to create applications, the task of imagining new and useful things for computers to do is a far from trivial one.

ACKNOWLEDGMENTS

This research was supported by Digital Equipment Corporation, the National Science Foundation (Grant No. IRI-8903034), Matsushita Electric Industrial Co., Boeing, Information Resources, Inc., Electronic Data Systems, Apple Computer Company, and the corporate members of the MIT International Financial Services Research Center.

We are indebted to Terry Winograd for suggesting the acronym "Oval" after hearing about the four components of our system and to John McDermott for suggesting the term "radically tailorable" as a description of the Oval system. The Oval system has benefited from the suggestions and work of numerous people over many years, including most recently: Mark Ackerman, Geoffrey Bock, Kevin Crowston, Troy Jordan, Jintae Lee, Charley Osborne, Dave Park, Mike Plusch, and Paul Resnick.

REFERENCES

1. Action Technologies, Inc. (1988). *The Coordinator, Version II, User's Guide*. Emeryville, CA:

2. Conklin, J. and Begeman, M. L. gIBIS: A hypertext tooling for exploratory policy discussion. In *Proceedings of CSCW '88 Conference on Computer-Supported Cooperative Work* (Portland, OR, Sept. 26-28, 1988), ACM Press, pp. 140–152.

3. diSessa, A. A. A principled design for an integrated computational environment. *Human Computer Interaction*. 1, (1985), 1–47.

4. Fischer, G. and Girgensohn, A. End-user modifiability in design environments. In *Proceedings of CHI '90 Conference on Human Factors in Computer Systems* (1990). ACM Press, pp. 183-191.

5. Galegher, J., Kraut, R. E. and Egido, C. (Eds.). *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*. Lawrence Erlbaum, Hillsdale, N. J., 1990.

6. Greenbaum, J. and Kyng, M. (Eds.). *Desgin at Work: Cooperative Design of Computer Systems*. Lawerence Erlbaum, Hillsdale, N. J., 1991.

7. Hutchins, E. L., Hollan, J. D. and Norman, D. A. Direct manipulation interfaces. In *User-Centered System Design*, D. Norman and S. Draper (Ed.), Lawrence Erlbaum, Hillsdale, N. J., 1986, pp. 87–124.

8. Kyng, M. Designing for cooperation: Cooperating in design. *Communications of the ACM*. 34, 12 (1991), 65-73.

9. Lai, K. Y., Malone, T. and Yu, K.-C. Object Lens: A spreadsheet for cooperative work. *ACM Transactions on Office Information Systems*. 6, 4 (1988), 332–353.

10. Lee, J. and Malone, T. W. Partially shared views: A scheme for communicating among groups that use different type hierarchies. *ACM Transactions on Information Systems*. 8, 1 (1990), 1–26.

11. Lotus. *Lotus Notes Users Guide*. Lotus Development Corp., Cambridge, MA, 1989.

12. Mackay, W. E. Patterns of sharing customizable software. In *Proceedings of CSCW '90 Conference on Computer-Supported Cooperative Work* (Los Angeles, CA, October 7-10, 1990), , pp. .

13. MacLean, A., Carter, K., Lovstrand, L. and Moran, T. User-tailorable systems: Pressing the issues with buttons. In *CHI '90 Conference on Human Factors in Computer Systems* (Seattle, WA, April 1-5, 1990), ACM Press, pp. 175-182.

14. Malone, T., Yu, K.-C. and Lee, J. (1989). *What good are semistructured objects? Adding semiformal structure to hypertext* (Technical report #102). Cambridge, MA: Center for Coordination Science, Massachusetts Institute of Technology.

15. Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R. and Rosenblitt, D. Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Information Systems*. 5, (1987), 115–131.

16. Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A. and Cohen, M. D. Intelligent information-sharing systems. *Communications of the ACM*. 30, (1987), 390–402.

17. Nardi, B. A. and Miller, J. R. An ethnographic study of distributed problem solving in spreadsheet development. In *Proceedings of CSCW '90 Conference on Computer-Supported Cooperative Work* (Los Angeles, CA, October 7-10, 1990), ACM Press, pp. .

18. Olson, G. M. and Olson, J. R. User-centered design of collaboration technology. *Journal of Organizational Computing*. 1, (1991), 61-83.

19. Schuler, D. and Namioka, A. (Eds.). *Participatory Design*. Lawrence Erlbaum, Hillsdale, N. J., 1992.

20. Sproull, L. and Kiesler, S. Computers, Networks, and Work. *Scientific American*. 265, 3 (1991), 84-91.

21. Winograd, T. A language/action perspective on the design of cooperative work. *Human Computer Interaction*. 3, (1987), 3–30.

22. Winograd, T. and Flores, F. *Understanding computers and cognition: A new foundation for design*. Ablex, Norwood, NJ, 1986.

23. Yourdon, E. *Modern Structured Analysis*. Yourdon, Englewood Cliffs, NJ, 1989.