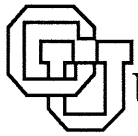
End-User Modifiability in a Water Management Application

Andreas C. Lemke, Stephen Gance

CU-CS-541-91 August 1991



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

University of Colorado at Boulder

Department of Computer Science

ECOT 7-7 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-7514, FAX: (303) 492-2844

October, 1990

End-User Modifiability in a Water Management Application

Andreas C. Lemke

Department of Computer Science and Institute of Cognitive Science
Engineering Center ECOT 7-7, University of Colorado, Boulder, CO 80309-0430
andreas@boulder.colorado.edu
Phone: 303-492-1503

Stephen Gance

Department of Computer Science and Center for Advanced Decision Support
in Water and Environmental Sciences (CADSWES)
Engineering Center ECOT 7-7, University of Colorado, Boulder, CO 80309-0430
stephen@boulder.colorado.edu
Phone: 303-492-3972

Submitted to the CHI'91 Conference
New Orleans, LA, April/May 1991

Abstract. We present principles for end-user modifiability and demonstrate them in ObSim, a river basin simulation system. End-user modifiability is supported by multiple computational mechanisms trading off expressivity for reduced complexity and domain-specificity to different degrees. The computational mechanisms used in ObSim include a construction kit and an object-oriented spreadsheet. We demonstrate the increased conceptual simplicity with a modification scenario.

Acknowledgements. We would like to thank Andreas Girgensohn and other members of the HCC community at Boulder for their insightful comments and support. We would like to thank as well Wayne Cheney of the US Bureau of Reclamation (USBR). This work was supported in part by grant No. IRI 8722792 of the National Science Foundation, cooperative agreement 8-FC-81-12480 between USBR and CADSWES, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo. Any opinions, findings, conclusions, or recommendations are the authors' and not necessarily the views of USBR or of the University of Colorado.

End-User Modifiability in a Water Management Application

Andreas C. Lemke

Department of Computer Science and Institute of Cognitive Science
Engineering Center ECOT 7-7, University of Colorado, Boulder, CO 80309-0430
andreas@boulder.colorado.edu

Stephen Gance

Department of Computer Science and Center for Advanced Decision Support
in Water and Environmental Sciences (CADSWES)
Engineering Center ECOT 7-7, University of Colorado, Boulder, CO 80309-0430
stephen@boulder.colorado.edu

Abstract

We present principles for end-user modifiability and demonstrate them in ObSim, a river basin simulation system. End-user modifiability is supported by multiple computational mechanisms trading off expressivity for reduced complexity and domain-specificity to different degrees. The computational mechanisms used in ObSim include a construction kit and an object-oriented spreadsheet. We demonstrate the increased conceptual simplicity with a modification scenario.

1. Introduction

Water resource management is a set of well-coordinated but technical interventions in the hydrological cycle undertaken to augment and better regulate the existing water supplies for meeting human needs more effectively (Saha, 1981). End-user modifiability is becoming increasingly necessary in water resource management because a great deal of new hydrological research is becoming available, water law is constantly being updated, new environmental realities are beginning to be understood, and the political climate changes. Current systems resist change because they lack a modular structure, and changing them requires extensive programming expertise.

It is highly desirable that users without extensive programming skills be capable of carrying out such changes. There are several obstacles counteracting this goal, however. The learning curve associated with learning how to modify a system is steep, and users quickly reach a plateau on which they no longer extend their abilities. The production paradox (Carroll & Rosson, 1987) prevents users from overcoming the steep learning curve. There are two approaches to solving this problem. One of them is to provide knowledge-based support systems to help users to learn and understand the complex in-

ternal structure of the system that is to be modified (e.g., Fischer & Girgensohn, 1990; Schoen, Smith & Buchanan, 1988). The second approach is to implement the system using an easily understood yet powerful computational paradigm and to maximally exploit the knowledge the users already have. Examples of such paradigms are graphical rewriting (Lewis, Rieman, & Bell, 1990), spreadsheets (Wilde & Lewis, 1990), and macro facilities. Both approaches are valid and typically should be combined to achieve the best possible effect. The present work focuses on the second approach. We explore a computational architecture that increases end-user modifiability in the domain of water resource management.

We first formulate principles for end-user modifiability that we have applied in the water management domain. We describe the limitations of current water management systems. Next we describe a new system, ObSim, illustrating the principles we formulated. We demonstrate the advantages of ObSim using a modification scenario and finally discuss some limitations of our approach.

2. Principles for End-User Modifiability

Formulation of principles as guidelines for interactive system design has been challenged by some researchers including Grudin (1989). While principles often fail to decide difficult tradeoff problems, they do provide useful guidance in many situations. In addition to general principles of interactive system design (e.g., *provide immediate feedback*), specific principles for end-user modifiability can be formulated. For instance, MacLean et al. (1990) advocate that a tailoring culture be created in a workplace. A tailoring culture is an attitude towards systems that includes *design* as an integral component. We propose the following principles for enhancing end-user modifiability.

Users control the domain-oriented aspects of the design. At least those changes addressing application domain issues are best done by the users rather than by computer specialists. The users are most familiar with their tasks, and they know more about the problem domain in which these tasks are accomplished. Computer specialists are not always available when needed and first have to be trained in the application domain to be able to understand and carry out the required changes. On the other hand, enabling domain specialists to make changes to computer science oriented aspects of the system is much harder because they lack the necessary knowledge.

Human problem domain communication. The design of systems for water resource management differs from the design of walk-up-and-use systems or systems such as word

processors and spreadsheet programs in that the users of the former systems are specialists in the water management domain. They are well trained and fluent in a jargon with its own technical terms and graphical conventions. We can exploit this fact by building into the system the important abstract operations and objects of the domain. Rather than communicating with *computers*, users should perceive their work as communication with *their domain of expertise*, in their natural idiom (Stelzner & Williams, 1988), and the computer should become effectively invisible. We call this the principle of *human problem domain communication* (Fischer & Lemke, 1988a). In an environment supporting human problem domain communication, designers build artifacts from application-oriented building blocks according to the principles of that domain—not the principles of computer software. Human problem-domain communication greatly simplifies the problem of forming a mental model of the system and, thus, is a big step towards end-user modifiability.

Multiple language levels. Designing for end-user modifiability requires choosing one or more computational paradigms in which to express the behavior and properties of the system. It is these computational paradigms that the users must manipulate to accomplish their modification goals. Two tradeoffs can be observed among the range of computational paradigms. The first tradeoff is that between domain-specificity and expressiveness: More domain-specific languages are typically less expressive. The second tradeoff is that between complexity and expressiveness: More expressive languages are typically more complex and harder to learn. The term *language* is used here in a general sense, including property sheets and other customization mechanisms. Languages at the general end of the spectrum gain their power from generic control and data structures that are basically all-encompassing but are removed from the concepts of the problem domain (opposite of human problem domain communication). Most systems provide one or two languages for modification: a high-level language with limited power and a general purpose programming language—the system implementation language—for all other changes. Additional languages at various positions on the complexity/expressiveness dimension are needed to smooth out the learning curve. We propose a design with multiple languages that form a stack of levels in which each higher level language can be compiled into, or is interpreted by, the next lower, more expressive language. Most of these languages should and can be domain-oriented, thus allowing human problem domain communication in end-user modification. Specialists in the domain of water management already use multiple levels of languages ranging from schematic diagrams to sophisticated mathematical models. Given a certain need for modification, the user must be able to select the language level appropriate for this modification.

Accessibility of the next lower level. Introducing more languages might at first seem counterproductive. With the multitude of languages comes an increased learning requirement. To mitigate this problem, users should be able to "see through" to the next lower level. This facilitates the transition to the next language level in several ways. Users can inspect the internals of existing constructs. Because they know the external behavior of the construct, they can learn the structures of the next lower level language of which the internals are constructed. Secondly, modifying and experimenting with those structures does not require a complete understanding and provides an easy entry point to the next language level. MacLean et al. (1990) report that a person who was not a Lisp programmer successfully modified a small piece of internal Lisp code to customize a "button."

3. Existing Systems

Water resource management is a complex domain requiring the cooperation of humans and computer systems.[1] There is a multitude of computer models in use today, but we focus on one of the most successful systems, MITSIM (MIT Simulation System; Strzepek, Garcia & Over, 1989). It was used on numerous river basins and is notable because it was one of the first to make the schematic breakdown of a basin explicit in the code thus increasing maintainability. Up to that time, the network model was not unknown but was not a key consideration in the decomposition of the model into (FORTRAN) code. MITSIM also attempted to be a general model in that data files for a particular basin can be input to the model so that it could be used in a variety of different basins. However, the description language was not expressive enough, and it was necessary to update MITSIM almost every time it was used in a new basin because of a need for different algorithms for certain aspects of the basin. Typically, the original code developer was the only one who could make the changes.

4. ObSim

In this section, we describe ObSim, an object-oriented modeling and simulation environment for water management based on the principles laid out in the previous section. ObSim supports end-user modifiability at multiple levels (Figure 1). We begin with a discussion of the lowest level: procedural programming language.

---

[1]For a task analysis see Gance (1990).

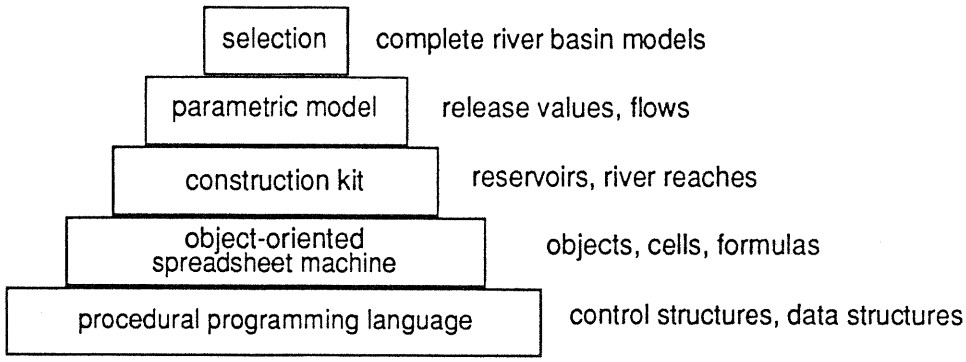


Figure 1: Language Levels for End User Modifiability in ObSim

---

4.1. Procedural Programming Language

The lowest interesting level of end-user modifiability in ObSim is the procedural programming language. In contrast to existing water management systems, ObSim implements the majority of its functionality—especially the domain-oriented constructs—not at this level but at the higher levels described below.

4.2. The Object-Oriented Spreadsheet Machine

The next level above the procedural programming language level is an object-oriented spreadsheet machine operating as a quantitative simulation system. The spreadsheet machine supports cells with values and formulae as well as automatic updating. However, unlike a regular spreadsheet in which cells are arranged in a two or three-dimensional array, in Obsim, a single column of cells is associated with basin objects (see below), and the cells are identified by name instead of by x and y coordinates. (For a discussion of related computational paradigms see Wilde & Lewis, 1990.)

Unlike the ASP object-oriented spreadsheet (Piersol, 1986) where objects are values of the cells, in Obsim, objects *are made up* of cells. Objects form a prototype inheritance hierarchy. Cells that are not locally defined in an object are inherited from the prototypes of that object. However, only the formula and not the value of a cell is actually inherited. The value of the cell is the result of computing the formula locally within the object. For instance (Figure 2), the usable capacity of a reservoir is defined as the total capacity minus the dead capacity minus the flood buffer. This is represented in the *reservoir* object, which is a prototype for other objects (called extensions) such as *lake-mead.* Each extension object inherits the formula for usable capacity from *reservoir* but has its own value, which is the result of evaluating the formula in the context of the extension, i.e., with the local values of *total-capacity*, *dead-capacity*, and *flood-buffer.*

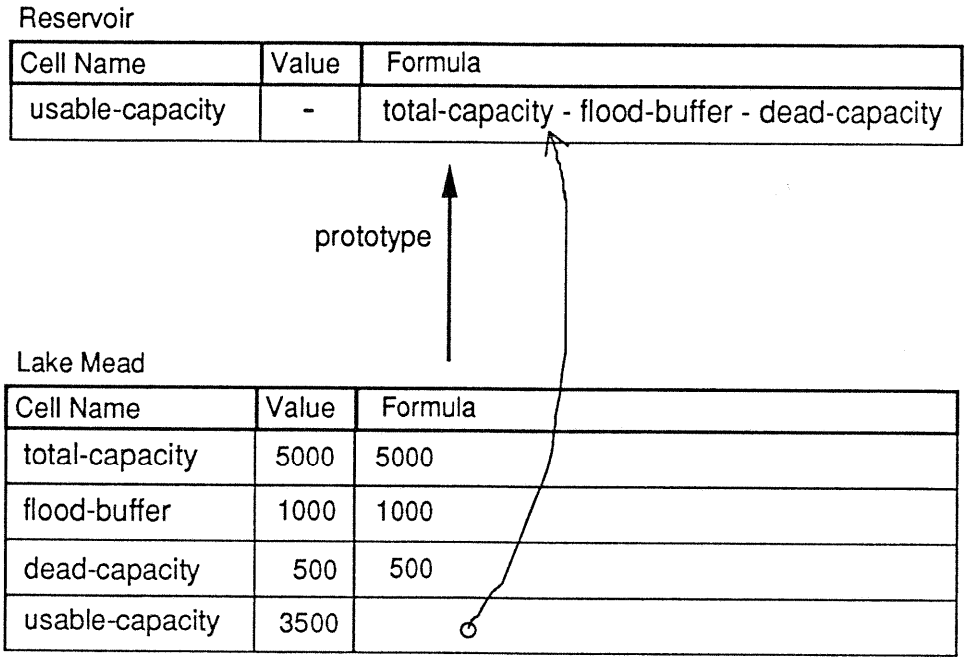


Figure 2: Inheritance in the Object-Oriented Spreadsheet Machine

---

This mechanism provides a simple, yet powerful, conceptual model. Water management workers are familiar with mathematical formulae. They can modify the behavior of a basin object by changing the formulae associated with the various cells or by adding and deleting cells. The automatic update facility provides immediate feedback on the screen whenever a value or formula is modified. The prototype inheritance model achieves great computational power with conceptual simplicity and has several advantages over the more popular class-instance model (see also Lieberman, 1986). First, there is no fundamental difference between classes and instances; any object can serve as a prototype, and users do not have to learn the differences between classes and instances. Second, instance variables do not have to be declared; they are created by being assigned to. Third, a new type of object can be created and debugged as an individual object without the additional (conceptual) overhead of maintaining both a class and an instance for testing the class. After debugging the new object, it can be used with no change as a prototype for an arbitrary number of extension objects. Fourth, behavior that deviates from the norm for a certain class of objects can be directly implemented in the object, without having to resort to such "tricks" as creating a private class for the special object. These advantages must be weighed against the disadvantages of prototype inheritance. For example, the performance of class-instance systems seems to be better, and the class-instance distinction may be viewed as better representing the fundamentally different nature of abstract and concrete objects.

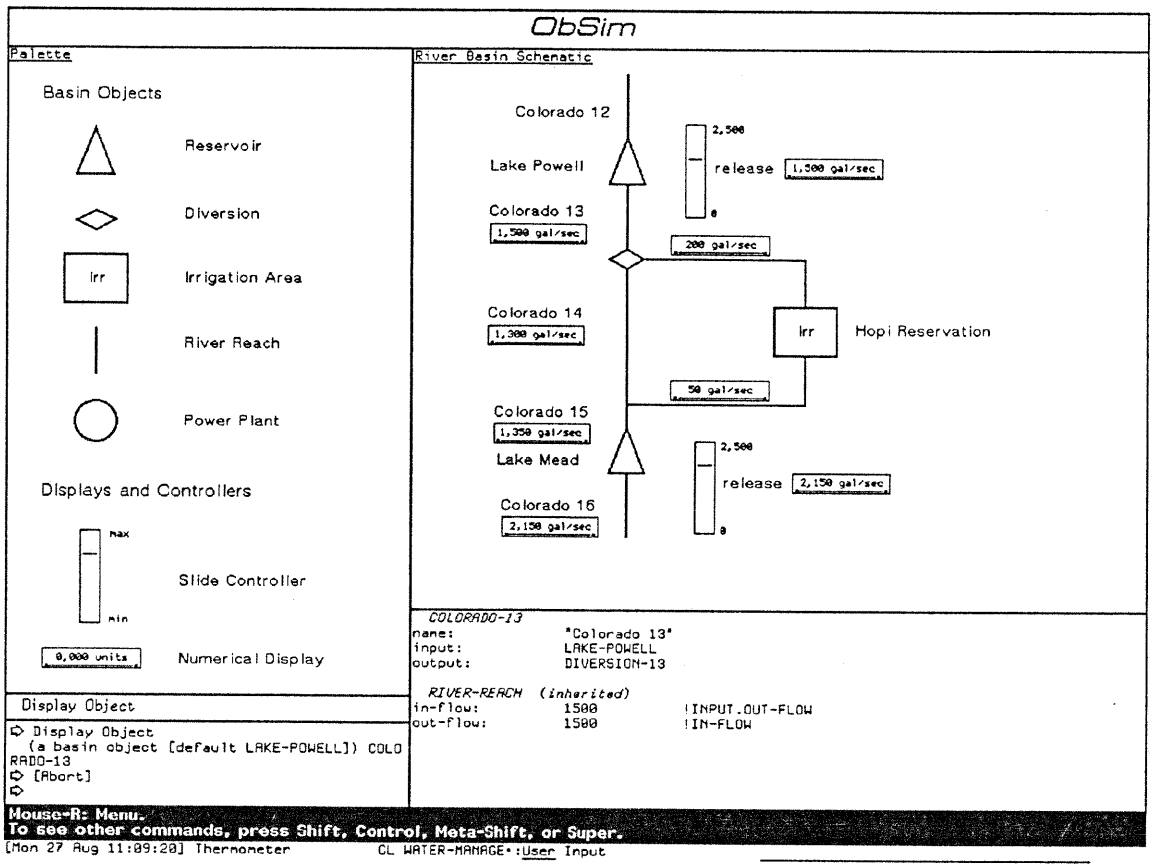


Figure 3: ObSim Screen Image

4.3. Basin Objects

The basin object level allows the user to graphically create, delete, and connect basin objects of different types (Figure 3). This level is a construction kit in the terminology of Fischer and Lemke (1988b). Prototypes for various basin objects, such as reservoirs, diversions, and river reaches, are displayed in a palette, and using those prototypes, users create and modify models of specific river basins in the river basin schematic window. End-user modifiability extends to the palette without requiring additional mechanisms. Prototype inheritance allows users to extend the palette by copying and modifying any basin object from the palette or from the river basin schematic thus overcoming a limitation of the construction kit concept: the inability to add new objects to the palette.

This level closely models the concepts of the domain of water management, and thus realizes our concept of human problem domain communication. The objects can be manipulated without knowing how they function internally at the spreadsheet machine level. Nevertheless, the spreadsheet machine level is always accessible to the user in the

bottom right window of Figure 3. The user can see how graphical operations are translated into corresponding operations at the spreadsheet machine level. Creating an object from the palette is translated into creating an extension of a prototype. Connecting one object to another is translated into assigning the name of the second object to the output cell of the first object and assigning the name of the first object to the input cell of the second object.

4.4. Parametric Model

At this level, users manipulate the release of water from reservoirs, the amount of water diverted for irrigation, and they feed in predicted weather data from the National Weather Service to produce 24-Month Studies, guideline documents handed down to the operators that do short term (i.e., daily) management. At this level, no new objects are created nor are connections between them changed. This level provides a parametric model of the basin. The user actions are limited to adjusting numerical parameters (e.g, water release values) and reading out computed quantities (e.g, water salinity and reservoir water levels). Obsim supports this level of modeling through *controllers* and *displays* on the screen that are attached to spreadsheet cells.

4.5. Selection

The simplest level of end-user modifiability is the selection of complete predesigned models of different grain-size and of different geographical areas. No specific support for retrieving an appropriate model (a catalog) is currently available (e.g., Nakakoji & Fischer, 1990).

5. Scenario

To illustrate typical end-user modification episodes, we describe the addition of a salinity[2] budget model to a river reach. Salinity problems are an example of a recent concern that was unknown or thought unimportant when the original simulation models were built. A simple salinity budget model (Narasimhan et al., 1980) is described by this formula:

$$t_{out} = t_{in} + t_{nat} + t_{ag} + t_{ps}$$

The salt loading at the end of the reach t_{out} is the sum of the loading at the beginning t_{in},

[2]Salinity refers to the amount of dissolved salts in the water.

Basin Object

Cell Name	Value	Formula
in-salinity	-	input.out-salinity
out-salinity	-	in-salinity

prototype

River Reach

Cell Name	Value	Formula
out-salinity	-	in-salinity + natural-salt-loading + agricultural-salt-loading + point-source-salt-loading
point-source-salt-loading	0	0

prototype

Reach 940

Cell Name	Value	Formula
in-salinity	2000	*(inherited from Basin Object)*
natural-salt-loading	120	120
agricultural-salt-loading	55	55
out-salinity	2175	*(inherited from River Reach)*

Figure 4: Modifying a Model to Account for Salinity

---

loading from natural (diffuse) sources t_{nat}, agricultural (diffuse) sources t_{ag}, and point sources t_{ps}. Since this modification involves adding some formulas, the spreadsheet machine level in our stack of languages seems to be appropriate. We define an *in-salinity* and *out-salinity* for each basin object (Figure 4). Using the strategy of specifying each formula at the most general level at which it applies, we conclude that the above formula should be specified at the *river-reach* level. Inheritance allows us to specify a default salinity for all basin objects in the *basin-object* prototype and a default loading from point sources at the *river-reach* level. This scenario illustrates our principle of human-problem domain communication and conceptual simplicity: There is a straightforward mapping from the salinity model found in the literature to its representation in ObSim. Users can accomplish this mapping with only a few simple strategies.

6. Limitations

The scope of end-user modifiability in ObSim is limited. Currently the graphical shape of the basin objects can not be influenced above the procedural programming language level. As mentioned, a catalog component must be added. Also, only simple visualization tools for simulation results are available.

7. Summary

We have presented principles for end-user modifiability. These principles have been illustrated in ObSim, a system for water resource management, which represents a significant advance over existing systems. ObSim makes multiple tradeoffs between conceptual complexity and expressivity by offering multiple computational paradigms including a construction kit and an object-oriented spreadsheet. Operations at each higher-level language are translated into corresponding operations at the next lower-level language. This paper lays out and argues for a (partial) theory of end-user modifiability grounded in a system implementation. We have not yet evaluated the theory empirically.

Acknowledgments. We would like to thank Andreas Girgensohn and other members of the HCC community at Boulder for their insightful comments and support. We would like to thank as well Wayne Cheney of the US Bureau of Reclamation (USBR). This work was supported in part by grant No. IRI 8722792 of the National Science Foundation, cooperative agreement 8-FC-81-12480 between USBR and CADSWES, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo. Any opinions, findings, conclusions, or recommendations are the authors' and not necessarily the views of USBR or of the University of Colorado.

References

Carroll, J. M. & Rosson, M. B. (1987). Paradox of the Active User: In Carroll, J. M. (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 80-111). Cambridge, MA: The MIT Press.

Fischer, G. & Girgensohn, A. (1990). End-User Modifiability in Design Environments. *Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA)*, 183-191. New York: ACM.

Fischer, G. & Lemke, A. C. (1988). Constrained Design Processes: Steps Towards Convivial Computing: In Guindon, R. (Ed.), *Cognitive Science and its Application for Human-Computer Interaction* (pp. 1-58). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fischer, G. & Lemke, A. C. (1988). Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication. *Human-Computer Interaction, 3*(3), 179-222.

Gance, S. (1990). *Human Problem-Domain Communication in River Basin Planning and Operations* (Technical Report). Boulder, CO: CADSWES, University of Colorado.

Grudin, J. (1989). The Case Against User Interface Consistency. *Communications of the ACM, 32*(10), 1164-1173.

Lewis, C. H., Rieman, J. & Bell, B. (1990). *Problem-Centered Design for Expressiveness and Facility in a Graphical Programming System* (Technical Report CS-CU-479-90). Boulder, CO: Department of Computer Science, University of Colorado.

Lieberman, H. (1986). Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. *OOPSLA'86 Conference Proceedings*, 214-223. New York: ACM.

Nakakoji, K. & Fischer, G. (1990). Catalog Explorer: Exploiting the Synergy of Integrated Design Environments. *Software Symposium'90 (Kyoto, Japan)*, 264-271.

Narasimhan, V. A., Huber, A. L., Riley, J. P. & Jurinak, J. J. (1980). *Development of Procedures to Evaluate Salinity Management Strategies in Irrigation Return Flows* (Technical Report UWRL/P-80/03). Logan, UT: Utah Water Research Laboratory, Utah State University.

Piersol, K. W. (1986). Object-Oriented Spreadsheets: The Analytic Spreadsheet Package. *OOPSLA'86 Conference Proceedings*, 385-390. New York: ACM.

Saha, S. K. (1981). River Basin Planning as a Field of Study: Design of a Course Structure for Practitioners: In Saha, D. K. & Barrow, C. J. (Eds.), *River Basin Planning*. New York: Wiley.

Schoen, E., Smith, R. G. & Buchanan, B. G. (1988). Design of Knowledge-Based Systems with a Knowledge-Based Assistant. *IEEE Transactions on Software Engineering, SE-14*(12).

Stelzner, M. & Williams, M. D. (1988). The Evolution of Interface Requirements for Expert Systems: In Hendler, J. A. (Ed.), *Expert Systems: The User Interface* (pp. 285-306). Norwood, NJ: Ablex Publishing Corporation.

Strzepek, K., Garcia, L. & Over, T. (1989). *MITSIM 2.1 River Basin Simulation Model, Vol. 1* (Technical Report). Boulder, CO: CADSWES, University of Colorado.

Wilde, N. & Lewis, C. H. (1990). Spreadsheet-Based Interactive Graphics: From Prototype to Tool. *Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA)*, 153-159. New York: ACM.