University of Colorado at Boulder

Department of Computer Science

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

KNOWLEDGE-BASED SPREADSHEETS

Gerhard Fischer and Christian Rathke
Department of Computer Science
Campus Box 430
University of Colorado,
Boulder, Colorado, 80309

In Proceedings of AAAI-88,
7th National Conference on Artificial Intelligence, St.Paul, MI, August 1988

Knowledge-Based Spreadsheets

Gerhard Fischer and Christian Rathke
Department of Computer Science and Institute of Cognitive Science
University of Colorado, Campus Box 430
Boulder, CO 80309

Abstract

Spreadsheet systems have changed the way the world perceives and deals with computers. In an attempt at maintaining the positive elements of spreadsheets while overcoming some of their limitations, we have developed FINANZ, a computational environment to develop financial planning systems. FINANZ contains a form-based user interface construction system, which allows the creation of advanced user interfaces without the need for conventional programming. It uses constraint based programming for the representation of knowledge about the application domain. Its layered architecture (based on object-oriented knowledge representation) supports the modification and extension of the system and the dynamic generation of explanations.

1. Introduction

If we believe that the real impact of the computer in the information age will be determined by whether domain experts and technologically unsophisticated users will perceive the computer as a useful and usable device -- then spreadsheet systems have changed the way that the world perceives and deals with computers.

In this paper we first describe dimensions of success models for user-centered computer systems, which provide some rationale for the success of spreadsheets and which help us to identify some of their shortcomings. To overcome these shortcomings, we have used methods and techniques from Artificial Intelligence to develop FINANZ, a computational environment to develop financial planning systems. The major contributions of FINANZ are illustrated and we conclude by evaluating our system building effort and by indicating extensions and future research in this area.

2. Success Models for User-Centered Computer Systems

One way to advance the state of the art in a field is to identify "success models", i.e., activities, systems, and tools which work well. Previously, this approach has provided us with a great deal of insight in our work in designing computer-based learning and working environments by looking at skiing as a success model [Fischer 81; Burton, Brown, Fischer 84]. In doing so, we have identified the features of success models and tried to transfer them to less successful systems. In the work described in this paper, we have taken a similar approach in the area of user-centered computer systems by focusing on spreadsheet-based programs. We see a strong mutual relationship between research in user-centered system design and artificial intelligence, especially if we consider the important goal of AI being to build

systems augmenting human intelligence (as interactive knowledge media, as tools for conversation, and as intelligent support systems, which support cooperative problem solving processes between humans and computers [Stefik 86]).

2.1 Dimensions of Success Models

Without any attempt to compile a complete list (see [Norman, Draper 86] for additional views and features of user-centered systems), we try to characterize the dimensions which explain why spreadsheets are success models, indicate the shortcomings of spreadsheets, and demonstrate the contributions of FINANZ.

Conviviality. According to Illich [Illich 73], "*convivial tools are those which give each person who uses them the greatest opportunity to enrich the environment with the fruits of his or her vision. Tools foster conviviality to the extent to which they can be easily used, by anybody, as often or as seldom as desired, for the accomplishment of a purpose chosen by the user.*" Currently most systems belong either to the class of *general purpose programming languages* or to the class of *turn-key systems.* General purpose programming languages are convivial in the sense that they allow "in principle" the user to do everything, but they are too far away from the conceptual structure of the problem and it takes too long to get a task solved. Turn-key systems are easy to use, but they can not be modified by the user and therefore they do not allow users to contribute to *their* goals.

Convivial systems (supporting modifiability, tailorability, and extensibility) are a necessity if we believe in the fundamental assumption that it is impossible for a system designer to create a problem domain-oriented environment capturing all functionality that might conceivably be needed for a given application.

Useful and Usable Systems. Useful computers which are not usable are of little help; but so are usable computers which are not useful. One of the major research goals of user-centered system design is to resolve this design trade-off and to achieve these two goals simultaneously [Fischer 87]. Useful computers require complex systems with a rich functionality (e.g., providing a large number of suitable abstractions) and are therefore in danger of becoming unusable. To make high functionality systems usable and to exploit their power, computer-based intelligent support systems are needed which take advantage of the interactive and dynamic nature of computer systems. *Usable* systems are often limited in their usefulness by their limited applicability and extensibility.

Subjective Computability. In user-centered system design the crucial issue is not what user can do "in principle" -- what matters is what users can *really* do. The epistemological adequacy of a formalism in user-centered system design is primarily not a formal or theoretical issue (theoretically almost all formalisms and programming languages are Turing-equivalent) but a cognitive issue. For many problems, the question of *subjective computability* (to create systems which are usable for tasks which many users were unable to tackle in the past) is more relevant than whether a problem is computable in theory. Subjective computability can be increased by eliminating prerequisite knowledge and skills and by raising the level of abstraction towards the expertise of the user. Constrained design processes (such as selection, simple combination, instantiation, etc.) which users can handle are of greater relevance than unconstrained design possibilities which are beyond their grasp.

Human Problem-Domain Communication. Most computer users are not interested in computers per se, but they want to use the computer to solve problems and to accomplish *their* tasks. *Human problem-*

domain communication [Fischer, Lemke 88] has as its goal to build the important abstract operations and objects of a given application area directly into the environment. This implies that the user can operate with personally meaningful abstractions. In most cases the semantics of a problem domain should not be eliminated by reducing the information to formulas in first-order logic or to general graphs. Understandability of systems can be increased by allowing the user to directly manipulate the concepts of an application.

Reducing Complexity. User-centered system design is a worthwhile goal because there is no "conservation law of complexity" [Simon 81], which requires that the complexity and usability of a system is a given constant. Complexity can be reduced by exploiting what people already know and what they are already familiar with, by using familiar representations (based on previous knowledge and analogous situations), by exploiting the strengths of human information processing and by designing "better" systems which exploit the unique possibilities of interactive computer systems (e.g., by generating custom-tailored and user-centered representations [Fischer 87]).

2.2 Spreadsheets as Success Models

Spreadsheets can be considered success models by the sheer fact that they have changed the way the world perceives computing. They have created a turn-around in buying consideration; users want a spreadsheet -- on which computer it would operate is only a secondary consideration ("software buys hardware"). The popularity and usefulness of spreadsheets is based on the fact that they make contributions to all of the criteria enumerated in the previous section: they let users do *their* tasks, they have turned out to be usable and useful by being able to handle a wide range of problems, they increase the subjective computability of non-programmers, they let domain experts operate effectively by matching their conceptualizations, and they reduce complexity through their value propagation mechanisms by eliminating the need to care about low level computations and consistency maintenance.

2.3 Shortcomings of Spreadsheets

Despite their success, spreadsheets have a number of limitations. They are not "smart" programs -- there is no underlying knowledge machinery to attach arbitrary complex daemons to individual fields (e.g., parsers for allowing input information be given in different notations or dependency relationships to allow the creation of dynamic explanations). They suffer from a lack of extensibility (despite the examples provided by [Kay 84]), which limits their applicability for problems which do not fit exactly the basic spreadsheet paradigm. The lack of extensibility is due to the fact that spreadsheet systems are constructed as monolithic systems instead of as layered systems using multiple levels of abstraction.

Spreadsheets do not support constraint-based computations, they only allow value propagation in one direction ("one-way constraints"). They cannot be extended in natural ways to more general form-based systems (e.g., the usability decreases when one has to deal with several spreadsheets simultaneously). Models of the user are not supported. They could be used to present different external representations and views of reduced complexity (e.g., in the case of a grant proposal (see next section) for the applicant or the program director in the granting agency).

3. FINANZ: Going Beyond Spreadsheets

FINANZ is a computational environment to develop financial planning systems that are based on an extension of the spreadsheet paradigm. It supports its users in various domains such as project budget planning. It gains its power by being tuned to very specific application domains, in which operations often are only meaningful to the domain expert. By building on a more powerful object-oriented base, designers are able to develop more powerful sets of functions than are found in spreadsheet programs. In the following sections, we describe FINANZ from the viewpoints of the user and the system designer.

3.1 Interacting with FINANZ

Spreadsheets have been successful because they adopted an interaction format that people were already familiar with, and has enhanced its functionality by making the entries change dynamically. With FINANZ we want to keep the basic interaction style and enhance it only by features that are implied by its increased functionality.

In most spreadsheet systems, there are two conceptually distinct modes of operation: programming and executing. In the programming phase, the dependency structure is established; in the execution phase values are supplied by the user and propagated to depended fields by the system.

In FINANZ, a value to a field is supplied by selecting it using the mouse. A formula is specified by selecting the operations from a menu that is associated with the field (Figure 3-1).

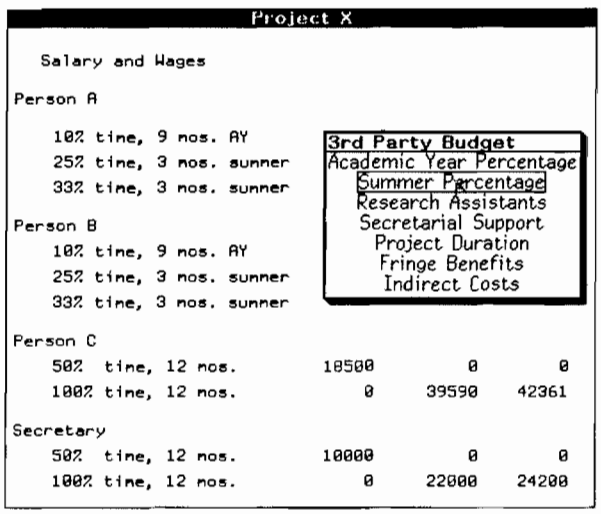

Figure 3-1: Formula Specifications

A formula is specified by selecting the appropriate operations from a menu. Operations are domain dependent and reflect the system's knowledge about the application domain. Entries such as "Indirect Costs" are applied to a field. The system guides the user by asking for the fields that contribute to the selected operation. The user specifies these fields by pointing at them with the mouse.

There are some differences on the interface level between FINANZ and spreadsheet systems:

- *Free positioning of fields.* Instead of having a predefined grid of fields, FINANZ cells are "liberated" [Lewis 87] in the sense that they can be put anywhere inside the forms' boundaries. Their sizes can also be changed to allow for longer pieces of text. New fields are generated by copying existing ones. They initially take the shape of their originals, but can then be modified using operations such as `move` and `reshape` that are generic to all screen objects.
- *Typed fields.* A field's content is an external representation of some data object. The user's input is interpreted and converted to a standardized internal representation. From there the possibly modified external representation is produced and displayed in the field. The interpretation and conversion processes are determined by the type of the field. This allows to connect sophisticated parsers with fields. For instance, a field containing dates accepts the date specification in a variety of syntactical forms. The printed representation of a date can depend on, for instance, the length of the field, the demands of the specific application, or the user's preference.
- *Multiple forms.* FINANZ is integrated in a window-based environment [Boecker, Fabian, Lemke 85] which supports the concurrent display and activation of multiple forms. Dependencies can be easily established between fields of different forms using direct manipulation.

The increased functionality provides new challenges to the user interface. In spreadsheets the value of a field is determined either by the user or by a formula. In FINANZ, a field can be part of any number of constraints. The user can ask for all the information that is needed to determine the rationale for a value of a certain field. The system displays the relationships and the user's input values that are responsible for a derived value (Figure 3-2). It is important to note that this includes non-static information. In contrast to spreadsheets, the dependency structure of a derived value of a FINANZ field is not predetermined. It depends on the previous dialogue and activations of value propagations.

Explanation capabilities become especially important when the user is asked to resolve a conflict that is generated by more than one constraint and several field values (see Section 3.2). FINANZ signals the conflict by highlighting all the responsible fields and displaying a message asking the user to take over control (Figure 3-3).

The system designer may have implemented some conflict resolution strategies such as preference of one field over an other. If there is only one field that is in conflict with the user's input, the value typed in by the user may be preferred. Users can influence conflict resolution by marking fields as constants. This has the effect that their value is fixed for the conflict resolution process. This technique allows users to explore effects of modifications under the condition that certain fields remain unchanged.

3.2 Designing FINANZ

In designing FINANZ, we combined the two main perspectives of user interface design in Artificial Intelligence: FINANZ is an interface using AI techniques and an interface to an AI system. In systems such as FINANZ which gain their power through domain-oriented communication capabilities, a strict separation between interface and application seems to be neither desirable nor possible.

The internal representational mechanisms of FINANZ are based on *constraints* [Borning 79; Steele 80]. By selecting an operation from the menu, users establish a constraint between fields. Constraints operate bi-directionally and they propagate changes automatically. Understanding the functionality of a spread-

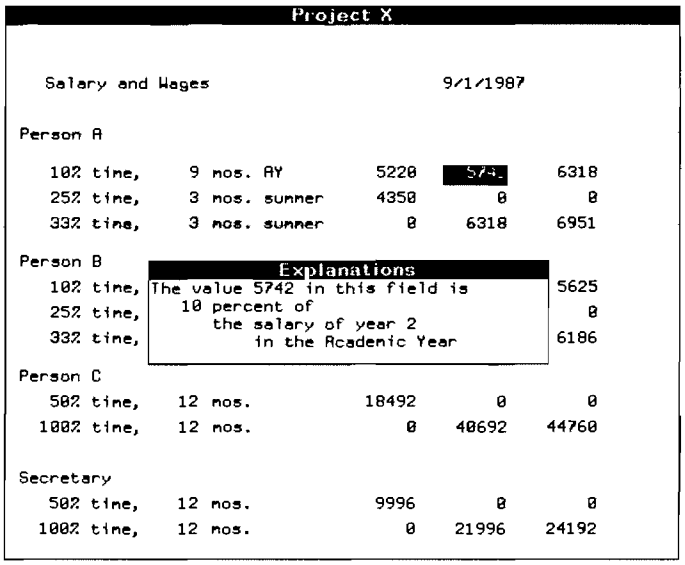

Figure 3-2: Explanations

The user has asked for an explanation of a field's contents. The system displays the fields and constraints that are ultimately responsible for the derived value. Constraints are verbalized in the explanation window, which is displayed as an answer to the explanation request. Explanations of constraints are augmented by field names and actual field values. Explanation capabilities are especially important when there is no obvious way by which a field value is determined.

sheet system in terms of constraints rather then in terms of operations allows not only for multi-directional propagation of values, but also for a better way of representing complex relationships. The computational paradigm is that of a constraint satisfaction process that takes all of the specifications into account.

In the representational basis of FINANZ constraints are special classes in OBJTALK, the object-oriented knowledge representation language [Rathke 86] which is used as implementation vehicle for FINANZ. When the user selects an operation from the menu the system instantiates the appropriate constraint class with the fields specified by the user. Changing one of these fields is internally represented as a message passing event to all constraint instances in which the field plays a role. As a result new values are computed and propagated to other fields. The computations are represented as methods in the constraints' classes. When a depended field is set, the sources are recorded with it, i.e., the method that computed the value along with the fields that triggered the method. This information is used for explanation purposes, for detecting reasons for conflicts, and dependency directed backtracking.

New constraints can be introduced by simple subclassing, because they are objects that define their behavior in classes of OBJTALK. Most of the properties such as recording reasons for derived values and producing explanations are located in a common superclass and need not be specified each time a new constraint class is introduced.

The described representational mechanisms enhance spreadsheet programming in various ways (these

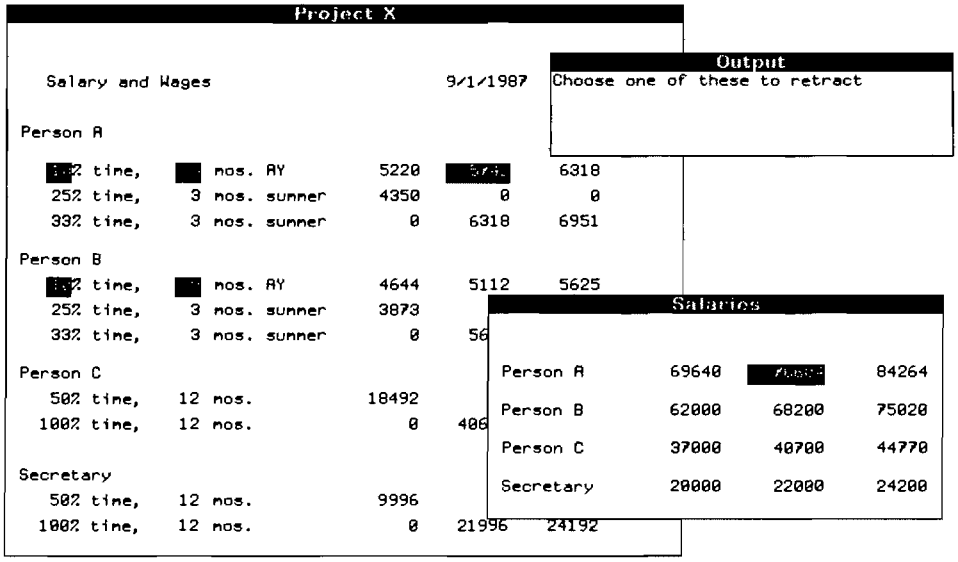

Figure 3-3: Conflict Resolution

The user is asked to resolve a conflict caused by several constraints and a number of fields. The amount assigned to "Person A" in the second year of the project is in conflict with his overall salary and the percentage and time values of the "Academic Year". Either of these values can be retracted to solve the conflict. The percentage and time values are also constraint to be equal to the ones of "Person B". Their modification is likely to affect "Person B's" figures. In this situation, users can ask for explanations of all of the field values, including those which are highlighted.

enhancements to spreadsheet technology are currently only available to system designers; users are unable to define new constraints):

- *Constraints can be non-numeric.* Dependencies between spreadsheet fields are represented as reactions to messages. These reactions are not restricted to perform numerical operations. Any kind of symbolic computation is possible. For instance, information about a person or an individual such as their qualifications and their status can be related to their salaries and/or their duties.
- *Fields can be involved in more than one constraint.* In spreadsheet systems, the contents of a depended field has to be determined by a single formula. In FINANZ, fields usually take part in more than one constraint. Field values can therefore be derived from multiple sources. This can cause conflicts if users provide more information than necessary to derive a field value. FINANZ provides several built-in conflict resolution strategies that, for instance, rank the user supplied inputs. If no conflict resolution strategy is specified, the system points out the conflicting fields and asks the users to specify which field they would like to be changed (see Figure 3-3).
- *Constraints can be combined.* Often, constraints that are specific to an application domain are combinations of more primitive constraints. The constraint for the time percentage of the working hours during the Academic Year, for instance, is constructed by combining the more primitive percentage and multiplication constraints.

The design methodology used in FINANZ is important. Moving from general purpose spreadsheets to specific application domains, it is crucial to develop FINANZ-like system employing high-level abstractions. In the same way as FINANZ provides the appropriate domain related abstractions for its users, the representational technology provides the appropriate abstractions to *construct* FINANZ-like systems.

This is achieved by a layered system architecture (Figure 3-4). A unique set of forms and fields defines the financial expertise for a specific project. Parameters are set by filling form fields with values. Inferences are drawn by the constraint propagation machinery. The system records dependencies, applies conflict resolution and generates explanations. A specific problem, which is given by a set of field values, is solved by a constraint satisfaction process.

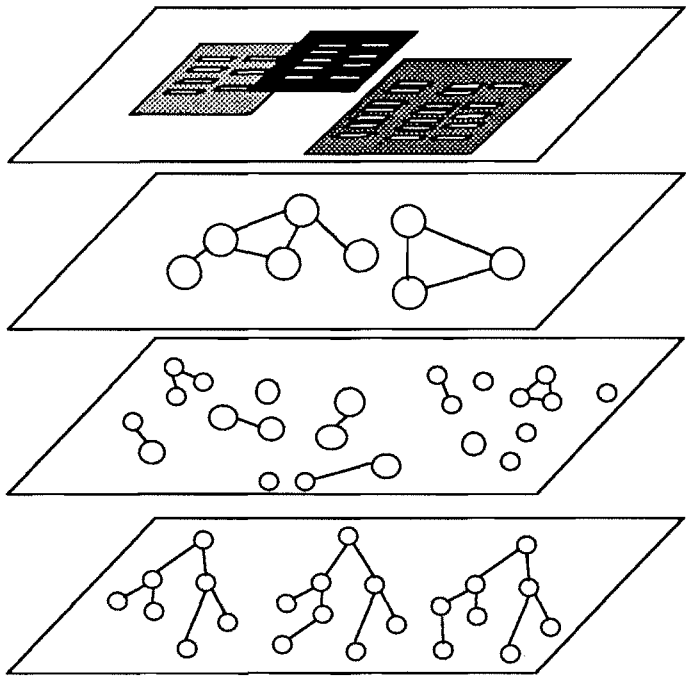

Figure 3-4: A Layered Architecture for FINANZ

The layered architecture of FINANZ allows to employ spreadsheet technology for bridging the gap between the two top most layers. Layers are problem specific at the top and general at the bottom. A modified combination of language primitives at the layer of primitive arithmetic constraints (3rd layer from the top) resulted in a financial planning system for a slightly different domain (see Figure 3-5).

Constraints are established during the programming phase. FINANZ becomes a meta-system for designing budgetary relationships. The important difference to other meta-systems (such as EMYCIN) is the level of abstraction that is used. FINANZ primitives are domain dependent. This makes them less general but at the same time more usable for the purpose for which they are designed (see Section 2.1). The gap that needs to be bridged from the primitives of the language to the intended result is much narrower than the one starting from general representational formalisms such as frames, rules, and constraints. By intentionally reducing the generality we are supporting the knowledge engineering task.

Each layer provides the basis for the generation of different layers of the next higher level. Users of FINANZ can design many budgetary systems involving different individuals, time spans and percentages. Designers can augment and modify domain related constraints to construct abstractions for a related

application domain. The most basic layer, that we are concerned with in this context, is represented by OBJTALK. Conceptual primitives such as defaults, restrictions, and rules are mapped into objects and classes that communicate by message passing.

The importance of the layered architecture for the rapid construction of a number of related system is demonstrated by a system developed some time ago using FINANZ. In this system the budget planning knowledge for the German Ministry for Research and Technology was represented (Figure 3-5).

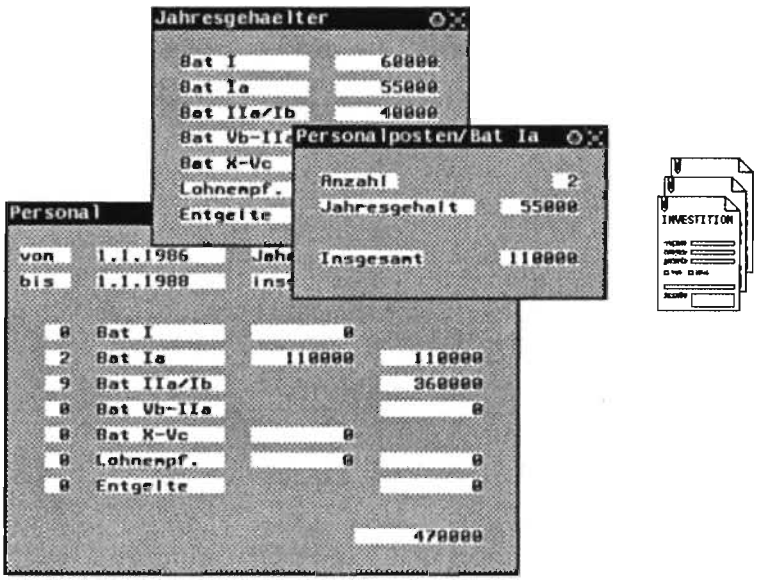

Figure 3-5: A Different Application Domain for FINANZ

Forms designed to support the financial planning process for projects with the German Ministry for Research and Technology.

4. Conclusions and Future Directions

Using more powerful representational mechanisms than in spreadsheets, we have to be careful not to loose those aspects which made spreadsheets a success model. Design tradeoffs are balanced in a different way in FINANZ than in spreadsheets. With FINANZ we have overcome some of the shortcomings of spreadsheets mentioned previously -- but have we introduced other ones? By providing more support for specific application domains, the tradeoff between generality and familiarity of concepts on one side and specialization on the other side has to be carefully evaluated. There is a strong interdependency between systems which support human problem domain communication, and the necessity for modifiability and tailorability of systems. FINANZ shows that an object-oriented approach towards knowledge representation [Rathke 86] and the use of a layered system architecture provides a good environment to make the construction of domain-oriented systems a practical and worthwhile activity. FINANZ as a major application

system has had a strong impact on the shape of our tools at the lower levels (see Figure 3-4) and has served as a major driving force for the continuous enhancement of our tools over the years.

To abandon general computational environments in favor of increased subjective computability raises the important question of what kind of general characteristics a problem must have to make spreadsheets or FINANZ a useful implementation technology. It is important to describe this space, so users can get a feeling of what kind of problems they can solve. Some of the extensions which we want to address in our future work on FINANZ are: to extend the number of abstractions used (e.g., to include abstractions from related domains such budget or tax law). The modifiability and tailorability of the system should be enhanced by providing a kit for the construction of new constraints at the end-user level (with the goal that the domain expert becomes even more independent of the knowledge engineer [Borning 86]).

Acknowledgments

The authors would like to thank Andi di Sessa, Hal Eden, Jonathan Grudin, Andreas Lemke, Clayton Lewis, Helga Nieper and Bill Swartout for criticizing drafts of this paper. The research was supported by grant No. DCR-8420944 from the National Science Foundation, and grant No. MDA903-86-C0143 from the Army Research Institute.

References

[Boecker, Fabian, Lemke 85]
H.-D. Boecker, F. Fabian Jr., A.C. Lemke, *WLisp: A Window Based Programming Environment for FranzLisp*, Proceedings of the First Pan Pacific Computer Conference, Australian Computer Society, Melbourne, Australia, September 1985, pp. 580-595.

[Borning 79]
A.H. Borning, *Thinglab -- A Constraint-Oriented Simulation Laboratory*, Technical Report SSL-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, 1979.

[Borning 86]
A.H. Borning, *Defining Constraints Graphically*, CHI'86 Conference Proceedings, ACM, Boston, Ma, April 1986, pp. 137-143.

[Burton, Brown, Fischer 84]
R.R. Burton, J.S. Brown, G. Fischer, *Analysis of Skiing as a Success Model of Instruction: Manipulating the Learning Environment to Enhance Skill Acquisition*, in B. Rogoff, J. Lave (eds.), *Everyday Cognition: Its Development in Social Context*, Harvard University Press, Cambridge, MA - London, 1984, pp. 139-150.

[Fischer 81]
G. Fischer, *Computational Models of Skill Acquisition Processes*, Computers in Education, Proceedings of the 3rd World Conference on Computers and Education (Lausanne, Switzerland), R. Lewis, D. Tagg (eds.), July 1981, pp. 477-481.

[Fischer 87]
G. Fischer, *Making Computers more Useful and more Usable*, Proceedings of the 2nd International Conference on Human-Computer Interaction (Honolulu, Hawaii), Elsevier Science Publishers, New York, August 1987.

[Fischer, Lemke 88]
G. Fischer, A.C. Lemke, *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 3, 1988.

[Illich 73]
I. Illich, *Tools for Conviviality*, Harper and Row, New York, 1973.

[Kay 84] A. Kay, *Computer Software*, Scientific American, Vol. 251, No. 3, September 1984, pp. 52-59.

[Lewis 87]
C.H. Lewis, *NoPumpG: Creating Interactive Graphics with Spreadsheet Machinery*, Technical Report CS-CU-372-87, Department of Computer Science, University of Colorado, Boulder, CO, August 1987.

[Norman, Draper 86]
D.A. Norman, S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Rathke 86]
C. Rathke, *ObjTalk: Repraesentation von Wissen in einer objektorientierten Sprache*, PhD Dissertation, Universitaet Stuttgart, Fakultaet fuer Mathematik und Informatik, 1986.

[Simon 81]
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Steele 80]
G.L. Steele, *The Definition and Implementation of a Computer Programming Language based on Constraints*, Technical Report MIT-TR 595, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1980.

[Stefik 86]
M.J. Stefik, *The Next Knowledge Medium*, AI Magazine, Vol. 7, No. 1, Spring 1986, pp. 34-46.