LiTH-IDA-R-88-14 ISSN-0281-4250

INSTITUTIONEN FÖR DATAVETENSKAP
UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

History, State and Future of User Interface Management Systems

by

Jonas Löwgren

INDUSTRISERIEN
ASLAB, May 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

LiTH-IDA-R-88-14 ISSN-0281-4250

History, State and Future of User Interface Management Systems

by

Jonas Löwgren

Abstract: This paper is an attempt to survey the topic of User Interface Management Systems (UIMSs). We give a short account of the historical development of UIMSs, try to capture what is today regarded as state of the art in the area, and examine the role of a UIMS in the process of software development. We also summarize several furure research directions commonly recognized as important, and finally give a short outline of our own proposed contribution to the field, addressing the issue of user interface management in knowledge-based systems.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

History, State and Future of User Interface Management Systems

Jonas Löwgren
Departement of Computer and Information Science
Linköping University
Linköping, Sweden

March 31, 1988

Abstract

This paper is an attempt to survey the topic of User Interface Management Systems (UIMSs). We give a short account of the historical development of UIMSs, try to capture what is today regarded as state of the art in the area, and examine the role of a UIMS in the process of software development. We also summarize several future research directions commonly recognized as important, and finally give a short outline of our own proposed contribution to the field, addressing the issue of user interface management in knowledge-based systems.

1 Introduction

Since 1982, when the term "User Interface Management System" first occurred, the number of people interested in that area has grown steadily. Hundreds of systems have been implemented and reported and as many relevant formal theories have been developed or rediscovered. The research work put into software tools for user interface management and design has increased at the same rate as the computerization of our every-day environments.

This paper is an attempt to survey the wide and prospering field of UIMSs. We start by trying to define and characterize what actually constitutes a UIMS. After that, we give a short historical survey, first in chronological form and then a more detailed outline of the development in a few areas central to UIMSs. This historical journey ends up in our present times, where we give an overview of the current state of the art in UIMS research and review several points that are currently disputed. There is also an account of what is today recognized as problems with UIMSs. The next section examines UIMSs as a part of the software development process, and particularily the use of rapid prototyping for user interface design. The final two sections are aimed towards the future of UIMSs. In section 6, the issues generally recognized as the most important for further research are surveyed, and in section 7 we give a short account of our own planned contribution to the field.

2 What is a UIMS?

Since there are many ideas of what User Interface Management Systems (UIMSs) really are, this paper would have a hard time in discussing them if we wouldn't start by establishing a common understanding. This section addresses the problem of defining and characterizing a UIMS in order to set the stage for the following discussions.

2.1 Definition of a UIMS

In the litterature, there are several operational definitions of a UIMS in terms of it capabilities as a tool. Hill [26] gives a definition that is attributed to Jim Foley:

- A tool used by a User Interface Administrator to build user interfaces for applications, much like a Data Base Administrator uses a Data Base Management System to deal with long term storage problems for applications.
- Something is a UIMS if it has something that determines the sequence of valid input (a parser or equivalent).

This definition attempts to explain the concept of a UIMS by referring to the (at the time, at least) more well known concept of a DBMS. Hill [26] then generalizes this definition to:

> A UIMS comprises special tools and techniques for implementing user interfaces.

This is undoubtedly a valid definition, but in the same time not very informative. Norman, Draper and Bannon [41] try to elaborate further on the tool aspect by explaining how a UIMS is used:

> A UIMS provides a way for a designer to specify the interface in a high-level language. The UIMS then translates that specification into a working interface, managing both the details of the display and its associated input and output and also the interaction with the rest of the program...

In [2], Betts et al proceed along the same lines to further refine this definition in terms of activities and purposes:

> A User Interface Management System (UIMS) is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and management (control) of the interaction in an application domain across varying devices, interaction techniques and user interface styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development. A UIMS can be viewed as a tool for increasing programmer productivity. In this way it is similar to a fourth generation language, where the concentration is on specification instead of coding.

To summarize, we have now arrived at an understanding of the function of a UIMS as well as its place in the software development process. We find that Hill's general definition, albeit not very informative, still is a good summary of what a UIMS *really* is.

2.2 Characteristics

According to our final definition above, a UIMS is to be used both in designing a user interface and in managing the user interaction in the application domain. This means that there are two categories of people faced with the UIMS: the *software developers* and the *end users*. (With the term "end user", we shall understand the person using the finished application.) Correspondingly, the UIMS can be described from two different viewpoints [2].

A software developer regards a UIMS as a tool that provides support for the definition of the user/application dialogue, imposes external control on the application, provides support for the presentation of the application's output and includes an interactive component providing support for the interaction between an application and an end user. From the software developer's point of view, a UIMS should provide a user interface with the following characteristics:

- Consistency
- Support for all kinds of users, from novice to expert
- Support for error handling and recovery

The points listed above are actually such that every user should demand of every application he uses. And why should software developers be an exception?

When the software developers use a UIMS, it encourages the development of better software by:

- Providing a consistent user interface between related applications
- Making it easier to change the user interface design when needed
- Encouraging development and use of reusable software components
- Insulating applications from the complexities of the environment
- Supporting ease of learning and use of applications

For the end user, the primary goal of a UIMS is to support the easy and effective use of an application. Even though end users need not be aware of the UIMS between them and the application, these are the advantages provided by a UIMS:

- Consistent user interface across applications ("house style")
- Support for multiple levels of help or assistance
- Support for training

- Support for end user tuning of the interface
- Support for extensibility of application

Most of the advantages above are due to the fact that the user interface is in some sense separated from the application. We will discuss this issue further in section 4.1. This separation also impacts the application development process, since the user interface design becomes a distinct part of the process. This effect is discussed in section 5.

3 The history of UIMSs

This section is devoted to a survey of the development of UIMSs. The first part is a short chronological overview. In the second part, the development is discussed in terms of several important issues in the design and construction of UIMSs.

3.1 Chronological survey

According to Hill [26], the term User Interface Management System (UIMS) was first used around 1982 by Jim Foley and Jim Thomas. But as far back as 1968, William Newman described a system called the Reaction Handler which had several of the properties today associated with UIMSs. There were also several other system reported before 1982 that today are counted as UIMSs. (Refer to Hill [28] for further information about these pioneer systems.)

But it was in 1982 that the research boom started. The basic concepts of a UIMS were articulated at a workshop on Graphical Input and Interaction Techniques, and already in the proceedings of SIGGRAPH'82 appeared a paper entitled "A user interface management system" by D Kasik.

November 1-3, 1983, a workshop was held in Seeheim, West Germany. The emphasis was mainly on models and structures. This was also the origin of the famous Seeheim model (fig 1). This model shows the logical components that "must appear in a UIMS" (Green [14], p.9). The *presentation* component of the model is responsible for the external presentation of the user interface. The *dialogue control* component defines the structure of the dialogue between the user and the application program, and the *application interface* model is a representation of the application from the viewpoint of the user interface. This model is

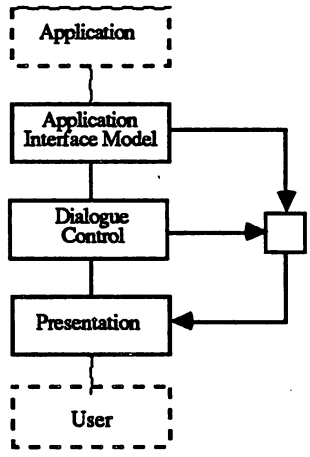

Figure 1: The Seeheim model.

often ascribed to Green, but a similar model was discussed by Edmonds [6] already in 1982.

In Seeheim, it was also realized that there were many hard problems concerning UIMS.

At the ACM SIGGRAPH Workshop on Software Tools for User Interface Management in Seattle, November 1986, the main points concerned what is actually a UIMS, and how it interfaces with the application and the environment. Here, the crucial question was also raised and debated: *Why aren't UIMSs commercially successful?* Another purpose of the meeting was to discuss directions of future research.

After these milestones on the road of UIMSs, development has proceeded further. What we now consider state of the art is discussed in section 4.

3.2 Aspects of development

The chronological overview in the previous section said almost nothing about the technical development of UIMSs. There are several areas within the UIMS realm that deserve our attention. In this section, we focus on two important areas: *dialogue specification techniques* and the *relation between UIMS and application.*

3.2.1 Dialogue specification

Recall Foley's original UIMS definition in section 2:

> ... Something is a UIMS if it has something that determines the sequence of valid input...

The "something" that Foley talks about is not specified. Norman, Draper and Bannon refined the criteria further:

> A UIMS provides a way for a designer to specify the interface in a high-level language...

One of the major points of using a UIMS is the ease to design, maintain and change the user interface under development. The urge to separate the user interface from the application implied a need for methods to specify various aspects of the user interface in a high-level, more declarative, manner. To identify these aspects, one appealing approach was to divide a user interface into *lexical*, *syntactical* and *semantical* components [34]. In this section, we now examine the development of specification methods for the syntactical aspect of the user interface — the dialogue structure.

To understand the development of dialogue specification techniques, it is necessary to be aware of the development in computer technology. In the first years of the 80s, the teletype interface with a keyboard and a monitor was the common environment for the user. This kind of interfaces inherently have a linear dialogue structure, where the user and the application take turns in acting. The graphical capabilities were limited, the typical dialogue styles were command language or form filling interfaces. Naturally, the UIMSs suggested comprised dialogue specification techniques reflecting this situation. The two major specification methods at this time were the *transition network* and its formal equivalent, the *grammar*.

The transition network is the oldest notation for dialogue specification. It dates back at least to 1968 and Newman's Reaction Handler. An example of a specification is found in figure 2. Each node in the network corresponds to a certain state in the dialogue, and the arcs between nodes define interactions, either input from the user or output from the system. In Jacob's formalism of figure 2, the input tokens are denoted by a lower-case 'i' as the first character of the name, which is otherwise in upper-case. The output token names are prefixed with an 'o' in the same way. The arcs labeled with lower-case names (like "login" and "setup") are non-terminals, which means that these arcs have associated networks that are invoked if the arc is reached. This idea of non-terminals is an attempt to address one of the major problems of transition networks: their managability. Transition networks are reported to be easier to read and understand than grammar representations [56], but they suffer from one serious problem: most interfaces have a large number of states, which implies a large and unwieldy network specification. To address this problem, the use of subnetworks has been suggested (see [14], but also figure 2: the use of non-terminals is an example of subnetworks). Other attempts to solve this problem include associating conditions with the arcs, and introducing memory of previous states [30]. There are many examples of UIMSs using the network specification technique, including the Rapid system of Wasserman and Shewmake [56], work by Jacob [34], and many more.

The grammar specification is equivalent in expressive power to the network, but tends to be harder to read and understand. An example of a grammar specification is found in figure 3.

```
<COMMAND> ::= PRINT <FILE> @ DOPRINT
          ! <FILE> := <FILE> @ DOCOPY

<FILE>    ::= <FILENAME> . <EXTENSION>
```

Figure 3: An attributed grammar specification. From Rönnqvist [48].

The grammar specification technique is most obviously applied to command language dialogues (cf [48]) but there are also examples of use for other kinds of dialogues. The SYNGRAPH system [42] manages fairly general graphical interaction with a grammar-based dialogue specification.

The network and the grammar specifications are both highly moded, and thus they worked well for the teletype interfaces of that time. But they were not particularly well suited for describing the graphical interfaces that became more and more common. These interfaces are of a more object-oriented character, and not explicitly moded. Although there were attempts to improve the existing techniques[1] , this was the signal for introducing a new specification technique: the *event model*. In 1983,

[1] See for instance [35], where Jacob extends the network specification to handle direct manipulation interfaces. Examples of quite recent systems adressing direct manipulation interfaces, but based on the older specification techniques, are the network-based HutWindows [36] or the grammar-based KES system [18] for CAD applications.

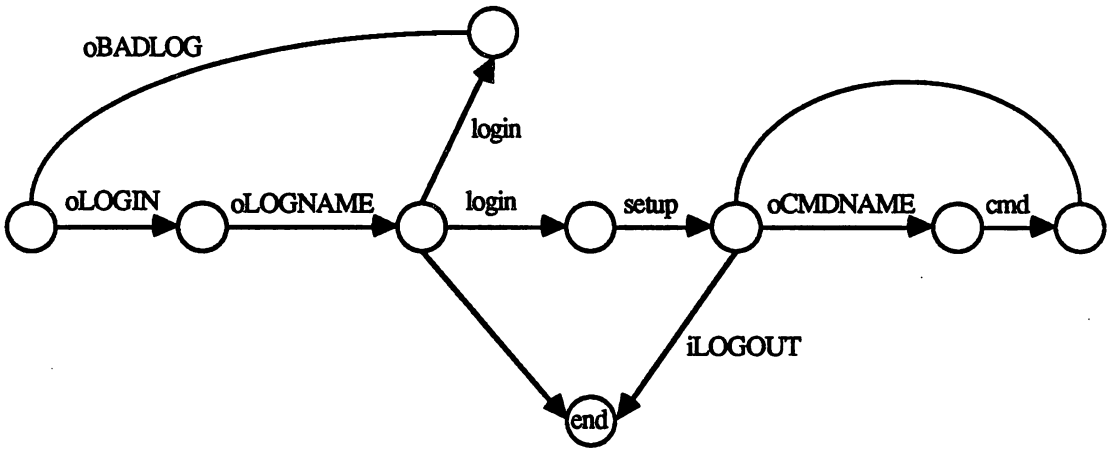

Figure 2: A transition network specification. From Jacob [34].

Mark Green [14] described the event model as follows:

> The event model views the user interface as a collection of events and event handlers. ... An event is generated each time the user interacts with an input device. These events are processed by the event handlers associated with the input or display device involved in the interaction. ... The collection of events processed by an event handler can be viewed as a state. The set of event handlers active (able to receive events) at any one time defines the legal user actions at that point in the dialogue.

An important difference [19] to the previous methods is that the transition network (and the grammar) incorporates *explicit* I/O event ordering, which means that the specification explicitly states what sequences of I/O events constitute a valid dialogue between user and application. The event model, on the other hand, features an *implicit* I/O event ordering, where the specification defines sets of I/O events without mentioning a specific ordering. The ordering restrictions are implicit in the UIMS that interprets the specification.

One of the main reasons for developing this new specification method was the issue of *feedback* in graphical user interfaces. The lowest level of graphical interaction, such as cursor tracking, could be handled uniformly at the lexical level of the transition network models, similar to character echoing or line wrapping in teletype interfaces. But if the designer wanted the user interface to provide other kinds of feedback, like the cursor changing shape when passing a certain object on the screen, it would be rather awkward [20]. Since this cursor changing is a state transition of a kind, the resulting transition network would be most complicated and large. It is also hard for transition network models to manage interface events that go on in parallel.

One of the first examples of a UIMS using the event model was the University of Alberta UIMS by Mark Green [15]. As in almost all UIMSs using this model, the event handlers were written in a kind of programming language. This procedural approach to specification was similar to object-oriented programming, and it was almost mandatory for some years. There are many more examples of UIMSs using the event model, including COUSIN by Hayes, Szekely and Lerner [19], Sassafras by Hill [24], IMAGES by Simoes and Marques [50] and many others. In the last years, there has been a quest for more declarative specification methods as an alternative to the programming-language approach of the original event model. The results will be discussed in section 4 of this paper.

To conclude, let us again point out the important observation that the development of dialogue specification methods is tightly coupled with the development of computer technology and in particular, the development of new interaction techniques.

3.2.2 UIMS–application relation

It is, of course, a great advantage to separate the user interface and its management from the application. The reasons are many, including maintainability, ease of design, reusability etc. The Seeheim model (figure 1) seemed at that time to be a perfect outline for a UIMS, with the lexical, syntactical and semantical aspects of the user interface represented in the presentation, dialogue control and application interface modules respectively [14]. This model was indeed a feasible approach for a UIMS addressing the teletype interfaces discussed in the previous section [29], but with the introduction of the direct manipulation interfaces the problems started. And the main problem of the Seeheim-type approach, which was aimed at separating all user interface aspects from the application, is the issue of *semantic feedback and performance.* This problem was indeed mentioned already at the Seeheim workshop by Strubbe [52].

For an example of semantic feedback (adapted from [5]), consider the Apple Macintosh. When the user drags a window on top of the trash can, no feedback other than the obvious of the window tracking the cursor is presented. But if the user instead drags a file icon on top of the trash can, the trash can is displayed inverted! In a sense, this is also lexical feedback, but nevertheless semantic knowledge is required to give it. And the problem is that this kind of feedback has to be virtually instant, which is probably impossible if there has to be tokens passed through three levels of abstraction. Because of the previous emphasis of separation of the user interface and the application, the communication between the two components is just too slow. What is needed is "fine-grain control" [38] rather than the "coarse-grain control" that is all that the application can exert in most existing UIMSs.

To sum up, the belief in the optimistic "separationist" approach represented by the Seeheim model seems to recede gradually. With the introduction of user interfaces requiring a lot of semantic feedback, the UIMS world has conceded to put back in the user interface something of the application that was removed in the first years of the 80s.

4 State of the art in UIMS research

In our brief journey through the history of UIMSs, we have seen a few areas that have been very important during the development towards the UIMSs of today. In this section, we will now survey what is regarded as state of the art in UIMSs today and then examine the issues that are today regarded as problems or points of discussion.

4.1 UIMSs today

In the historical survey, we saw many examples of UIMSs supporting the *conversational style* interfaces of teletypes. But the interaction metaphor that is currently growing fast is the *direct manipulation style.* The major part of UIMS research today is oriented towards making the systems support this metaphor. (The term "direct manipulation" was coined by Shneiderman [49]. In [33], Hutchins, Hollan and Norman gives a more abstract, user-oriented treatment of the direct manipulation concept.)

In direct manipulation, the user communicates with the *individual objects of interest* rather than with the system as a whole. This implies for UIMSs that *syntax should be in terms of individual objects.*

To increase the feeling of engagement [33], the issue of feedback is critical. Acting on objects in the application domain almost always has its consequences. Allowing the screen representation of these objects to immediately reflect these consequences greatly adds to the illusion that the representation of the object *is* the object. The issue of fast semantic feedback mentioned in the previous section is again central.

Hudson [32] includes the following conclusions about UIMS support for direct manipulation:

- Syntax should be expressed in terms of individual objects rather than the system as a whole. It should also be minimized, using physical actions such as pointing and dragging in preference to more syntactic concepts.
- Feedback, especially semantic feedback, is extremely important and needs better support.
- Flexibility in the presentation component is important; particularly the ability to

design specific interaction techniques and combine these into abstract devices.

- An application interface based on shared objects is preferable to conventional application interface based on semantic action routines.

All the above points influence the components of the Seeheim model (figure 1). It is nowadays recognized that this model is not adequate for direct manipulation interfaces, and several modifications are suggested. In Dance et al [5], the model in figure 4 is proposed. The

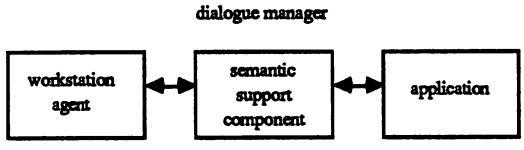

Figure 4: Relationship between system components. From Dance et al [5].

presentation component of the Seeheim model corresponds to the workstation agent and possibly some of the dialogue manager. The *dialogue control* and *application interface model* are similar to the dialogue manager and semantic support components, respectively. The main purpose of the dialogue manager is to provide a higher level of abstraction for interaction services. The purpose of the semantic support component is among other things to provide information for semantic operations such as feedback, default values and error checking and to provide application-specific help information.

All of Hudson's requirements above have been more or less investigated. Below is a short account of some interesting areas of research today.

The issue of syntax based on individual objects has a strong relation to *object-oriented programming*. Many UIMSs has recently been implemented using object-oriented techniques (often Smalltalk). Some examples are a system by Grossman and Ege [17], HIGGENS by Hudson and King [31] and IMAGES by Simoes and Marques [50]. There are also many systems using object-oriented techniques for direct manipulation interaction that would be regarded as graphical toolkits rather than true UIMSs. Some examples of this category are Fischer's WLisp [7] and Ida by Young [58]. Related examples of graphical toolkits, specifically intended for creating control panels, can be found in [8] and [21]. A formal treatment of the concept of presentation within an object-oriented framework can be found in [53].

One issue related to the object-oriented approach is the question of procedural vs. declarative information. One technique which has recently received much interest is the one of *constraints*. A constraint is a relation that must be maintained. Maintaining that relation is up to the underlying system instead of being the responsibility of the user. Constraints allow a declarative description of a user interface, since with constraints the designer specifies *what* relations are to hold, leaving it to the underlying system to decide *how* to maintain the relations. Borning and Duisberg's Thinglab [4] relies heavily on constraints for specifying the user interface. Another, more limited example of constraints is Juno by Nelson [40]. This is a system for graphics design where geometrical constraints are used.

Another area of interest is *specification of the dialogue syntax*. Traditionally, the specifications were constructed in some kind of language, either as a textual specification (like grammars or event handlers) or as a graphical description (like transition networks). But with the current ideas of minimizing the syntax and basing it on individual objects, other alternatives has been proposed. Some of the more notable are Myers' Peridot [38] where the general interaction techniques desired are inferred from user-given examples, and MIKE by Olsen [43] where the user specifies the semantic commands and all syntax is handled by defaults.

The paradigm of interacting with individual objects instead of with the system as a whole, has initiated much research on *concurrency* and *multi-threaded dialogues*. Tanner [55] recognizes the difference between *multi-device input*, where each user input stream comes from a different input device, *multi-thread input*, where each stream is processed in a manner that is lexically and syntactically separate from other streams, and *multi-tasking support*, where each stream is logically asynchronous with respect to the other. Multi-threaded dialogues are discussed by Hill in [25], and the subject of multi-tasking is addressed in [37]. Hill's UIMS, Sassafras [24], supports concurrent input from different input devices in a multi-threaded way. He gives an example of a two-handed interface (to a paint program) constructed in Sassafras. Flecchia and Berg-

eron has constructed a UIMS [9], containing a language called ALGAE that supports multi-threaded input. There are also many other examples. It is interesting to note that even the transition network model has been extended to handle some concurrency [35] although it does not support truly parallel input.

Another point of interest is providing support for *new interaction techniques.* Some examples are Rhyne's work on gestural interfaces[2] [49] and the possibilities of a touch tablet pointed out by Hill and Buxton [23].

4.2 Current problems and points in dispute

UIMSs have now gained acceptance in the research and business communities. The development of the area is, as we saw in the previous section, prospering. But despite all the advantages that UIMSs undoubtedly offer, there still remains the disturbing fact that *UIMSs are not widely available or used.* In this section, we examine the reasons for, and the ongoing discussion about, this.

Myers [39] summarizes the problem in the following way: there is no general acceptance for UIMSs because they are

- too hard to use,
- too limited in the types of interfaces that they can create, and
- not portable with respect to different machines, operating systems and graphics systems.

Green [16] emphasizes the portability aspect, in that he points out that compatibility is needed to database systems. Olsen [44] adds to the list the point of *maintainability,* which is crucial to commercial acceptance. Let us review the current discussion about each of these four points.

Usability

One of the points that determine how easy the UIMS is to use for user interface designers is the issue of *required programming skill.* The UIMS encourages viewing user interface design as a separate activity in its own right, and thus it can not safely be assumed that all user interface designers are programmers. Therefore, a UIMS that needs programming is probably harder to use than one that doesn't. Figure 5 shows a classification of systems for creating user interfaces by how much programming skills the user interface designer needs to know to use them. Peridot [38], which appears to the right on the axis, and therefore is assumed the easiest to use, is very demonstrational. The user uses direct manipulation techniques to create direct manipulation interfaces by giving examples.

Hill [27] takes a different position on this issue, in that he claims that the goal should not be to eliminate programming, but to provide the user interface designer with an appropriate set of tools. These tools could require the use of some simple programming skills. He shows example interfaces developed in Sassafras, where all programming constructs needed were simple arithmetic operators, conditional branching and function calls. It seems likely that also non-programmers could quickly learn to deal with these constructs. The advantages with a limited-programming requirement would be greater generality and simpler testing.

So a UIMS where limited programming skill is required gives greater generality and expressive power. The question is whether it is as easy to use as a totally programming-free system. The constraint-based approach described earlier may be a good way to combine flexibility and ease to use.

Generality

As we pointed out in a previous section, the development of UIMSs is tightly coupled to the development of new interaction techniques. A problem with most existing UIMSs is that they do not well enough support the creation of direct manipulation interfaces. Some issues that must be addressed by a UIMS capable of handling direct manipulation interfaces satisfyingly are [39]

- support for different types of interaction techniques,
- the ability to create new and different types of interaction techniques,
- using an appropriate, easy-to-use language for programming the parts that cannot be easily specified graphically,
- allowing multiple interaction techniques to be *available* at the same time,
- allowing multiple interaction techniques to be *operating* at the same time, and

[2] A gestural interface is the electronical analogue of paper and pencil, and the user interacts with the system via strokes of a stylus on a tablet.

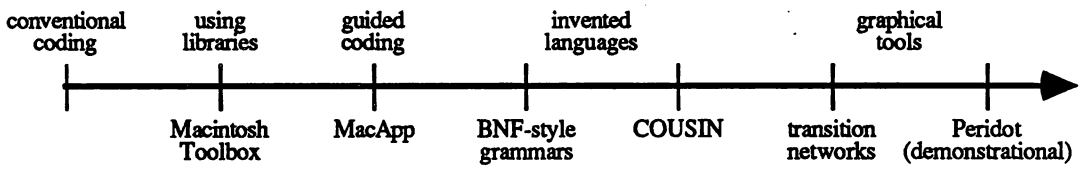

Figure 5: Some systems classified by programming skills needed. From Myers [39].

- providing semantic feedback, semantic error checking and semantic defaults that work at a sufficient speed.

Making UIMSs support direct manipulation interfaces is one of the most pursued research areas today, and in the research community solutions are already proposed to many of the above issues (see previous section). The problem is rather a matter of gaining acceptance for these solutions, and this is heavily dependent on the portability and availability of the systems.

Portability
Almost every UIMS that exists today is tightly tied to a particular operating system, computer and graphics package. This is because there are no appropriate high level interfaces that can shield the UIMS from having to directly manage the various input devices such as mouse and keyboard and from directly creating pictures on the screen. To make UIMSs more portable, powerful and abstract models must be created for input as well as for output. These models must then be integrated and made standard (*de facto* or *de jure*).

Another aspect of portability is that in a large application, the user interface is only part of the application program. When designing the application, the designers probably need to use a number of other tools like a database management package, a communication package or a distributed computing package [16]. If the notations for these different packages could be integrated, the development process would be considerably facilitated.

Maintainability
When a large program is built with a UIMS, most of the program's control structure is buried in the UIMS which to the application programmer is a black box. This renders the maintenance of the system a large problem. There are, in fact, commercial UIMSs advertising that the generated transition tables can be edited directly [44], which is somewhat like editing the object code of a production program. There can also be large problems due to the structure of the dialogue being interdependent with the side-effect behavior of the semantic actions.

As UIMSs develop, entirely new forms of program expression will be created. But the software engineering models commonly used were constructed for text oriented source files. The problems of software management must be readdressed within the context of a UIMS.

5 Using UIMSs for designing user interfaces

In this section, we view the UIMS as a tool in the process of software development, with special attention to the process of designing the user interface.

5.1 The value of UIMSs in user interface design

The process of developing a user interface, if viewed at a high level of abstraction, follows the standard waterfall life cycle of software development [2]. This is illustrated in figure 6. This model, though, may be hard to uphold when user interface tools are concerned, because it is hard to separate the specification, preliminary design and detailed design phases.

A user interface tool like a UIMS could influence all of the phases of the waterfall model. The *requirements* and the *specification and design* phases are all facilitated by the capability to create prototypes faster than an actual implementation. In the *implementation* phase, the process could benefit greatly from reusable software components. The UIMS could also provide support for the *testing* phase by means

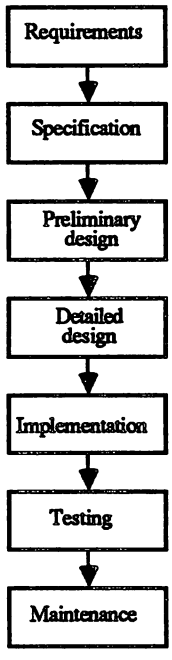

Figure 6: The waterfall life cycle of software development.

of logging and analysis of interactions (see section 6.2). *Maintenance* of the application user interface is considerably helped by the separation between user interface and the application. In the following section, we discuss the issue of prototyping in the specification and design phase in greater detail.

5.2 User interface design by prototyping

Even though there are claims that the endusers should be kept out of user interface design [57], most indications emphasize the importance of enduser involvment in the design process. The design of user interfaces should in this case be an iterative process, as illustrated in figure 7. There are many reports of extremely successful user interfaces developed using this approach, like the Xerox Star [51] and the 1984 Olympic Message System [13].

One thing that is characteristic for the iterative design approach is the use of *user interface prototypes* for involving the user at an earlier stage of the development process. Several systems, like Freburger's RAPID [12], have been developed just to enhance the production of prototypes. And according to Tanner and Buxton [54], a UIMS can help a great deal in devising the user interface prototypes of figure 7, mainly for the following reasons:

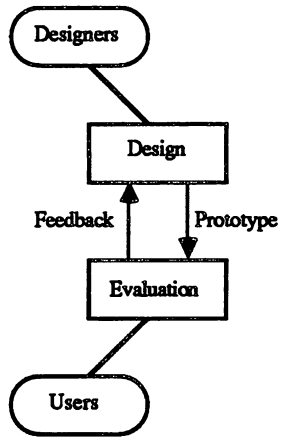

Figure 7: The iterative design approach (term from [28]).

- the dialogue portion of an application can (to some extent) be isolated from the functionality,
- the ideal UIMS should render all dialogue styles equally accessible, and
- the UIMS will render complex interfaces more maintainable.

But even simpler tools like paper mockups and "slide shows" may be enough to bring the design a long way towards a better user interface [3].

But prototyping can not be viewed just as a nice side effect of using a user interface tool for design. Riddle [47] points out a few points needing an advancement for truly successful prototyping:

- environments supporting the orderly creation and evolution of successive versions using piece parts,
- a measurement technology allowing truly scientific assessment of prototypes, and
- new development methods that accomodate the building of prototypes.

Alavi [1] has found that designers using prototyping may experience difficulties in managing and controlling the design process, even though the communication between designers and users is facilitated during the process.

This section may well be concluded with the words of Christiane Floyd [10]:

> Prototyping is, in itself, not a method for system development. It does not prescribe a sequence of steps which guarantee that an operational system satisfying all requirements are derived from fuzzy user concepts and attitudes.

UIMSs can contribute a lot to the development of user interfaces, but not by themselves. The entire development process and methods needs reconsidering with UIMSs in mind, in order to truly benefit from the advantages offered.

6 Future directions in UIMS research

Already in previous sections, we have hinted on what is today regarded as problems to address in future research. In this section, we will now examine some of the issues proposed for research. We divide these issues in three major groups. First, we have questions concerning the UIMS as a part of the software development environment. Secondly, we discuss proposed extensions to functionality provided by UIMSs today. Finally, we examine some proposed issues connected to the target application domain.

6.1 Software development

Existing UIMSs are based on detailed descriptions of the user interface to be constructed. These descriptions cover screen layout, dialogue structure and so on, and tend to be at a fairly detailed level. The designer must often specify the position of menues, and the order of menu items. The dialogue structure must be specified in detail. In some ways, dialogue specification has not improved significantly since 1968 (see section 3.2.1).

Most existing UIMSs tend also to be hard to learn. This can be inferred from the size of the manuals for commercial products [16], such as the Apple Macintosh.

One solution to the above problems would be to use a *higher level representation* of the user interface to be constructed. Green [16] observes that the three pieces of information required to build a user interface are

- a description of the problem that the user is trying to solve (a task model),
- a description of the functionality of the application routines, and
- the properties of the potential users. This includes the manual dexterity of the users, typing abilities, level of computer expertise, preferred interaction modes, and many other factors.

Green then proposes that we start developing a new generation of UIMSs that directly produce the user interface given the task model, the application functionality and the user properties.

Most UIMSs today are black boxes to the rest of the software development tools used in developing a large application (see section 4.2), which may imply maintenance problems. Another problem is that user interfaces and databases, who have a lot in common [16], are impossible to integrate today because of different notations. If UIMSs could be methodologically integrated in the software development process, they would significantly contribute to the development of even better application systems.

6.2 UIMS functionality

A UIMS plays a major part in encouraging an iterative approach to the design of user interfaces. The design and prototype production is generally well supported. But a UIMS could also be extended to support the *evaluation* phase. The evaluation could occur either before or after the user interface is implemented [16]. Tanner and Buxton [54] suggests that the UIMS should produce a protocol when the user is interacting with the implemented user interface. This protocol contains a time stamped record of each interaction between the user and the user interface. This protocol would then be analyzed by an automatic evaluation tool [54]. Some UIMS already produce protocols, but the automated analysis is a major research problem. Tools that could analyze the protocol with respect to time required to perform tasks, number and types of errors and so on would be most helpful.

The evaluation of a user interface before it is implemented is more difficult. There are techniques developed for evaluating user interfaces given formal descriptions of them. Foley's notation for the conceptual and semantic aspects of a user interface [11] is an example of such a description. It is conceivable that knowledge-based evaluation techniques can be useful in capturing more of the heuristic knowledge. An example of a knowledge-based tool, capturing heuristic knowledge of user interface design, is Fischer's WLisp [7].

6.3 Application domains

The user interface paradigm of *direct manipulation* is growing more and more dominant compared to the conversational paradigm. There is a need for UIMSs better adapted to direct manipulative interfaces (refer to the section 4.1). Among particularly vital issues are *semantic feedback* and *flexibility of presentation.*

Many of the directions proposed to address this need also make the UIMSs more complex [32], and UIMSs that are hard to learn are not used despite the advantages they might offer [43]. Currently we are facing a choice between systems which are very easy to use but produce simple or stylized interfaces, or systems that are complex to use but can support better interfaces. What is needed is a synthesis of these choices [32].

6.4 Summary of research issues

The research in the field of UIMSs is active and manyfold. The following is a summary of the issues discussed earlier. They are at the moment generally recognized as important, and attract a lot of research interest.

- The UIMS in the software development process
  - High-level systems for automatic user interface design based on high-level descriptions.
  - Regarding the UIMS as an integrated tool in the software development process.
- UIMS functionality
  - Automatic evaluation of user interface design.
- Target application domain issues
  - Better support for direct manipulation interfaces.
  - Find trade-off between power and ease to use.

There are of course other issues of a more fundamental importance. One example is the problem of finding a UIMS architecture that allows an *efficient implementation* of an application with a UIMS. If the performance of an application is unacceptable, the issues mentioned above are of no interest.

6.5 Our future research

The main research interest of our group is in the area of knowledge-based systems. Expert systems today often suffer from serious problems when the user interface is concerned. One example is the dialogue structure, which is often governed by the reasoning mechanism. This can give the effect that the user is required to answer several (perhaps seemingly unrelated) questions, and is then presented a solution to his problem. The explanation mechanisms are also often rudimentary. We feel that the situation of a user interacting with an expert system should be viewed more like the process of an expert solving a problem for somebody, communicating his knowledge and trying to explain his solving process. Our goal, thus, will be to understand this process and try to find out what is relevant for user interfaces to knowledge-based systems. This will further imply an investigation of user interface modelling for knowledge-based systems. A related issue is knowledge-based support for designing and evaluating user interfaces.

7 References

[1] Alavi, M. **An Assessment of the Prototyping Approach to Information Systems Development.** *Comm. of the ACM 27:6*, 1984.

[2] Betts, B, Burlingame, D, Fischer, G, Foley, J, Green, M, Kasik, D, Kerr, S, Olsen, D, Thomas, J. **Goals and Objectives for User Interface Software.** *Computer Graphics 21:2*, 1987.

[3] Bødker, S. **Through the Interface — a Human Activity Approach to User Interface Design.** Lic. Thesis DAIMI PB – 224, Aarhus University, 1987.

[4] Borning, A, Duisberg, R. **Constraint-Based Tools for Building User Interfaces.** *ACM Trans. on Graphics 5:4*, 1986.

[5] Dance, J, Granor, T, Hill, R, Hudson, S, Meads, J, Myers, B, Schulert, A. **The Run-Time Structure of UIMS-Supported Applications.** *Computer Graphics 21:2*, 1987.

[6] Edmonds, E. **The Man-Computer Interface — a Note on Concepts and Design.** *Int. Journal of Man-Machine Studies*, vol 16, p 231, 1982.

[7] Fischer, G. **An Object-oriented Construction and Tool Kit for Human-Computer Interaction.** *Computer Graphics 21:2*, 1987.

[8] Fisher, G, Joy, K. **A Control Panel Interface for Graphics and Image Processing Applications.** *Proceedings of CHI+GI'87*, 1987.

[9] Flecchia, M, Bergeron, D. **Specifying Complex Dialogs in ALGAE.** *Proceedings of CHI+GI'87*, 1987.

[10] Floyd, C. **A Systematic Look at Prototyping.** In Budde et al (eds) *Approaches to Prototyping*, Springer Verlag, Berlin, 1984.

[11] Foley, J. **Transformations on a Formal Specification of User-Computer Interfaces.** *Computer Graphics 21:2*, 1987.

[12] Freburger, K. **RAPID: Prototyping Control Panel Interfaces.** in OOPSLA'87 Proceedings *(SIGPLAN Notices 22:12)*, 1987.

[13] Gould, J, Boies, S, Levy, S, Richard, J, Schonard, J. **The 1984 Olympic Message System: A Test of Behavioral Principles of System Design.** *Comm. of the ACM 30:9*, September 1987.

[14] Green, M. **Report on Dialogue Specification Tools.** In Pfaff, G (ed) *User Interface Management Systems*, Springer Verlag, Berlin, 1985.

[15] Green, M. **The University of Alberta User Interface Management System.** *Computer Graphics 19:3*, 1985.

[16] Green, M. **Directions for User Interface Management Systems research.** *Computer Graphics 21:2*, 1987.

[17] Grossman, M, Ege, R. **Logical Composition of Object-Oriented Interfaces.** OOPSLA'87 Proceedings *(SIGPLAN Notices 22:12)*, 1987.

[18] Hammer, K, Hardin, J, Timmerman, T, Radin, D, Rhyne, J. **Automating the Generation of Interactive Interfaces.** 23rd Design Automation Conference, IEEE, 1986.

[19] Hayes, P, Szekely, P, Lerner, R. **Design Alternatives for User Interface Management Systems Based on Experience with COUSIN.** *CHI'85 Proceedings*, 1985.

[20] Hayes, P. **Executable Interface Definitions Using Form-Based Interface Abstractions.** In Hartston, H (ed) *Advances in Human-Computer Interaction*, Ablex Publishing Corp, NJ, 1985.

[21] Helfman, J. **Panther: A Specification System for Graphical Controls.** *Proceedings of CHI+GI'87*, 1987.

[23] Hill, R, Buxton, W. **Concurrency, Touch Tablets and Reusable Input Devices in Human-Computer Interaction.**

[24] Hill, R. **Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction—The Sassafras UIMS.** *ACM Trans. on Graphics 5:3*, 1986.

[25] Hill, R. **Event-Response Systems — A Technique for Specifying Multi-Threaded Dialogues.** *Proceedings of CHI+GI'87*, 1987.

[26] Hill, R. **Introduction to User Interface Management Systems.** Presented at Effektiv Interaktion Människa-Dator, Linköping, 1987.

[27] Hill, R. **Some Important Features and Issues in User Interface Management Systems.** *Computer Graphics 21:2*, 1987.

[28] Hill, R. **Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction.** PhD thesis, University of Toronto, 1987.

[29] Hill, R. **UIMS — Advanced Issues.** Presented at University of Linköping, February 1988.

[30] Hoppe, H, Tauber, M, Ziegler, J. **A Survey of Models and Formal Description Methods in HCI with Example Applications.** Report B.3.2a, ESPRIT project 385 HUFIT, 1986.

[31] Hudson, S, King, R. **A Generator of Direct Manipulation Office Systems.** *ACM Trans. on Office Information Systems 4:2*, 1986.

[32] Hudson, S. **UIMS Support for Direct Manipulation Interfaces.** *Computer Graphics 21:2*, 1987.

[33] Hutchins, E, Hollan, J, Norman, D. **Direct Manipulation Interfaces.** In Norman, D, Draper, S (eds) *User Centered System Design*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[34] Jacob, R. **An Executable Specification Technique for Describing Human-Computer Interaction.** In Hartston, H (ed) *Advances in Human-Computer Interaction*, Ablex Publishing Corp, NJ, 1985.

[35] Jacob, R. **A Specification Language for Direct-Manipulation User Interfaces.** *ACM Trans. on Graphics 5:4*, 1986.

[36] Koivunen, M-R, Mantyla, M. **HutWindows: An Improved Architecture for a User Interface Management System.** *IEEE Computer Graphics & Applications*, January 1988.

[37] Lantz, K. **Multi-Process Structuring of User Interface Software.** *Computer Graphics 21:2*, 1987.

[38] Myers, B. **Creating Interaction Techniques by Demonstration.** *IEEE Computer Graphics & Applications*, September 1987.

[39] Myers, B. **Gaining General Acceptance for UIMS.** *Computer Graphics 21:2*, 1987.

[40] Nelson, G. **Juno, a constraint-based graphics system.** *Computer Graphics 19:3*, 1985.

[41] Norman, D, Draper, S, Bannon, L. **Glossary.** In Norman, D, Draper, S (eds) *User Centered System Design*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[42] Olsen, D, Dempsey, E. **SYNGRAPH: A Graphic User Interface Generator.** *Computer Graphics 17:3*, 1983.

[43] Olsen, D. **MIKE: The Menu Interaction Kontrol Environment.** *ACM Trans. on Graphics 5:4*, 1986.

[44] Olsen, D. **Larger Issues in User Interface Management.** *Computer Graphics 21:2*, 1987.

[45] Olsen, D, Green, M, Lantz, K, Schulert, A, Sibert, J. **Whither (or wither) UIMS?** *Proceedings of CHI+GI'87*, 1987.

[46] Rhyne, J. **Dialogue Management for Gestural Interfaces.** *Computer Graphics 21:2*, 1987.

[47] Riddle, W. **Advancing the State of the Art in Software System Prototyping.** In Budde et al (eds) *Approaches to Prototyping*, Springer Verlag, Berlin, 1984.

[48] Rönnquist, R. **Customizing Command Language Dialogues with the YAKI package.** ASLAB MEMO 84-03, University of Linköping, 1984.

[49] Shneiderman, B. **Direct Manipulation: a step beyond programming languages.** *IEEE Computer 16:8*, 1983.

[50] Simoes, L, Marques, J. **IMAGES - An Object Oriented UIMS.** *Proceedings of INTERACT'87*, Elsevier Science Publishers B V, 1987.

[51] Smith, D, Irby, C, Kimball, R, Verplank, B. **Designing the Star User Interface.** *BYTE Magazine*, April 1982.

[52] Strubbe, H. **Report on Role, Model, Structure and Construction of a UIMS.** In Pfaff, G (ed) *User Interface Management Systems*, Springer Verlag, Berlin, 1985.

[53] Szekely, P. **Modular Implementation of Presentations.** *Proceedings of CHI+GI'87*, 1987.

[54] Tanner, P, Buxton, W. **Some Issues in Future User Interface Management System (UIMS) Development.** In Pfaff, G (ed) *User Interface Management Systems*, Springer Verlag, Berlin, 1985.

[55] Tanner, P. **Multi-Thread Input.** *Computer Graphics 21:2*, 1987.

[56] Wasserman, A, Shewmake, D. **The Role of Prototypes in the User Software Engineering (USE) Methodology.** In Hartston, H (ed) *Advances in Human-Computer Interaction*, Ablex Publishing Corp, NJ, 1985.

[57] Whiteside, J, Wixon, D. **Developmental Theory as a Framework for Studying Human-Computer Interaction.** In Hartson, H (ed) *Advances in Human-Computer Interaction*, Ablex Publishing Corp, NJ, 1985.

[58] Young, R. **An Object-Oriented Framework for Interactive Data Graphics.** OOPSLA'87 Proceedings *(SIGPLAN Notices 22:12)*, 1987.

I LiTH-IDA-R-88-14 **Jonas Löwgren:** History, State and Future of User Interface Management Systems

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-14

Upplagans storlek:
Number of copies:

Datum:
Date: May 1988

Projekt:
Project:

Titel:
Title: History, State and Future of User Interface Management Systems

Författare:
Author Jonas Löwgren

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

This paper is an attempt to survey the topic of User Interface Management Systems (UIMSs). We give a short account of the historical development of UIMSs, try to capture what is today regarded as state of the art in the area, and examine the role of a UIMS in the process of software development. We also summarize several furure research directions commonly recognized as important, and finally give a short outline of our own proposed contribution to the field, addressing the issue of user interface management in knowledge-based systems.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

IDA - institutionen för datavetenskap
vid Universitetet och Tekniska högskolan i Linköping

Tidigare rapporter i industriserien:

1982 **Erik Sandewall:** Ny teknik i kontorsdatasystem. (LiTH-MAT-R-82-17)
(Föredrag vid Nord-Data 82 i Göteborg, juni 1982.)

Sture Hägglund: Informationshantering i det elektroniska kontoret. (LiTH-MAT-R-82-27, ingår även i rapportserien från LIBLAB)
(Föredrag vid den 5:e Nordiska IoD-konferensen, Trondheim, juni 1982)

Anders Haraldsson: INTERLISP - en avancerad integrerad programmeringsomgivning för LISP-språket. (LiTH-MAT-R-82-29)
(Föredrag vid Nord-Data 82 i Göteborg, juni 1982.)

Hans Grunditz, Uwe Hein, Erik Tengvald: Artificiell intelligens i framtidens CAD/CAM-system (LiTH-MAT-R-82-32)
(Föredrag vid de nordiska CAD/CAM-dagarna, Göteborg, november 1982)

Software Systems Research Center. Progress Report 1982.
(Översikt över forskningsprojekt, personal och publikationer vid Datalogicentrum, LiTH.)

1981 **Ola Strömfors, Lennart Jonesjö:** The Implementation and Experiences of a Structure-Oriented Text Editor. (LiTH-MAT-R-81-03)
samt
Ola Strömfors: ED3 - Användarhandledning.

Sture Hägglund m fl: 80-talets elektroniska kontor: Erfarenheter från LOIS-projektet. (LiTH-MAT-R-81-04)

Peter Fritzson: Distribuerad PATHCAL: Förslag till ett distribuerat interaktivt programmeringssystem för PASCAL. (LiTH-MAT-R-81-05)

Dan Strömberg: Datorn - hjälpreda eller hot i det lilla företaget? (LiTH-MAT-R-81-06)

Software Systems Research Center. Progress Report 1981.

1980 **Kenth Ericson, Hans Lunell:** Redskap för kompilatorframställning (LiTH-MAT-R-80-39)

Software Systems Research Center. Preliminary Report 1980.

1979 **Arne Börtemark, Hans Lunell:** Implementering av Pascal på minimaldatorn: en tillbakablick. (LiTH-MAT-R-79-3)

Arne Börtemark: Felbekämpningsmedel, en första översikt. (LiTH-MAT-R-79-10)

Jerker Wilander: An interactive programming system for Pascal. (LiTH-MAT-R-79-37)

1978 **Kenth Ericson:** Pascal i Sverige. (LiTH-MAT-R-78-8) --- *Slut* ---

Jerker Wilander: Interaktiv programutveckling i Pascal - Programmeringssystemet Pathcal. (LiTH-MAT-R-78-20)

Erik Sandewall: Programmeringsteknik för flexibilitet. (LiTH-MAT-R-78-21)

Erik Sandewall m.fl.: The Linköping Office Information System (LOIS). An Overview of Facilities and Design. (LiTH-MAT-R-78-22)

Claes Strömberg, Henrik Sörensen: Beskrivning av fontsystemet och erfarenheter vid systemutvecklingen. (LiTH-MAT-R-78-23)

1977 **Harold W. Lawson, Jr.:** Future Trends for Mini-computer Systems. (LiTH-MAT-R-77-4)

1976 **Sture Hägglund, Östen Oskarsson:** En teknik för utformning av användardialoger i interaktiva datasystem. (LiTH-MAT-R-76-13)

IDA - institutionen för datavetenskap vid Universitetet och Tekniska högskolan i Linköping

omfattar ämnena *administrativ databehandling, datalogi*, och *telesystem*, med undervisning främst inom civilingenjörsutbildningen, datavetenskaplig linje och systemvetenskaplig linje. Inom IDA finns en bred forskningsverksamhet och ett forskarutbildningsprogram som leder fram till licentiat- och/eller doktors-examen. Institutionens forskning bedrivs f.n. i nio stycken grupper (laboratorier), som bl.a. arbetar inom områdena algoritmkomplexitetsteori, AI omgivningar, applikationssystem, CAD för elektronik, biblioteks- och informationssystem, logikprogrammering, databehandling av naturligt språk, programmeringsmiljöer, kunskapsrepresentation i logik samt i administrativ databehandling.

Industriserien

är en serie forskningsrapporter som särskilt vänder sig till den som arbetar med program- och maskin-varufrågor i industriell miljö eller inom den offentliga sektorns databehandling. I serien ingår dels rapporter från forskningsprojekt i Linköping som är av särskilt intresse för praktiker, dels översikter över aktuell forskning i världen i övrigt. Dessutom ingår presentationer av målsättning och strategi för institutionens forskning.

Hittills har utkommit:

1988 **Jonas Löwgren:** History, State and Future of User Interface Management Systems. (LiTH-IDA-R-88-14)

1987 **Arne Jönsson:** Naturligt språk för användardialog och databasförfrågningar. En sammanfattning av ett föredrag vid konferensen *Effektiv människa - dator-interaktion*, Linköping, 18-19 augusti 1987. (LiTH-IDA-R-87-25)

1985 **Harold W. Lawson, Jr.:** Sabbatical Report. (LiTH-IDA-R-85-05)

Sture Hägglund, Henrik Nordin, Roland Rehmnert, Kristian Sandahl: Utveckling av kunskapsbaserade applikationssystem i samverkan högskola - näringsliv (LiTH-IDA-R-85-09)

Harold W. Lawson Jr.: Swedish Participation in the Malaysian National Microelectronics Programme Also in *Proc. of ERSA (Economic Relations with Southeast Asia) Symposium*, Stockholm, October 21-25, 1985. (LiTH-IDA-R-85-11)

1984 **Bengt Lennartsson:** Programvarumiljöer. Produktionsteknik för programvara i Ada och andra språk. (LiTH-IDA-R-84-01)

Michael Pääbo: CAD-elektronik idag och i framtiden. (LiTH-IDA-R-84-03)

Erik Sandewall: Fjärde generationens Programvaruutbildning. (LiTH-IDA-R-84-13)

1983 **Michael Pääbo:** Introduktion till datorgrafik. (LiTH-IDA-R-83-02)

Mats Lenngren et al.: Datorgrafikdagar 7-9 juni 1983. (LiTH-IDA-R-83-05)

Pär Emanuelson: Programtransformationer (LiTH-IDA-R-83-06)

Sture Hägglund: Kunskapsbaserade expertsystem. Ny teknik för applikationsutveckling i nästa generations programvarusystem. (LiTH-IDA-R-83-07)

Erik Sandewall: Datavetenskaplig utvecklingsmiljö och kunskapsöverföringsprogram. (LiTH-IDA-R-83-10)

(*Fortsättning på pärmens insida.*)