Impact of user demands on knowledge-based systems

Autor(en): **Andersen, Tom / Carlsen, Niels Vejrup**

Objekttyp: **Article**

Zeitschrift: **IABSE reports = Rapports AIPC = IVBH Berichte**

Band (Jahr): **68 (1993)**

PDF erstellt am: **21.05.2024**

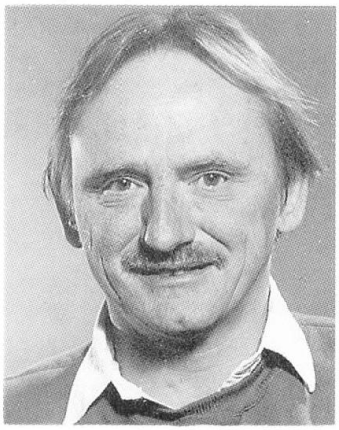
Persistenter Link: https://doi.org/10.5169/seals-51845

Impact of User Demands on Knowledge-Based Systems

Impact des exigences des utilisateurs sur les banques de données

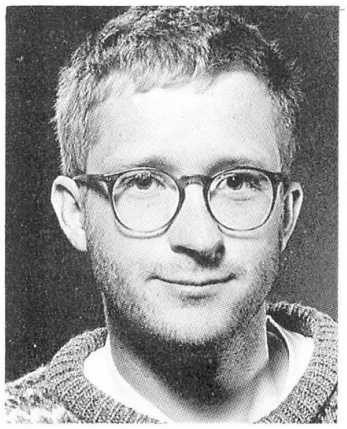
Einfluss der Benützeranforderungen an Entwurfsdatenbanken

Tom ANDERSEN
Assistant Professor
Techn.University
Lyngby, Denmark



Born 1954, M.Sc (1979) in Civil Eng., Technical Univ. of Denmark. Since 1989 member of staff, CAD Initiative. Research interests: knowledge-based systems, design theory, integration of software applications.

Niels Vejrup CARLSEN
Post doc
Techn.University
Lyngby, Denmark



Born 1963, M.Sc (1987) and PhD degree (1993) in Computer Science at the Technical Univ. of Denmark. Research interests: modelling of user interface software, user interface design, CAAD and virtual reality.

SUMMARY

When developing knowledge-based support systems for building design a query driven approach is generally considered very efficient. It provides a simple framework for knowledge acquisition and guides the modelling of the resulting knowledge-base. The identified queries define the structure of the user interface which relieves the designer from many considerations. However, this linkage gives certain system maintenance problems. Other issues in the design of knowledge-based design support systems arise from user demands such as flexibility of user interface and the existence of engineering strategies for logically separating the user interface software from the knowledge-base.

RÉSUMÉ

Le développement des banques de données basées sur les connaissances en matière de projet, implique une approche qui, partant des demandes des utilisateurs, est considérée efficace. Elle fournit un cadre à l'acquisition des connaissances et dirige la modélisation de la banque de données en résultant. Les demandes identifiées définissent la structure de l'interface et évitent un grand nombre de réflexions. Cette liaison soulève certains problèmes d'entretien. D'autres points de vue découlent des désirs de l'utilisateur, comme p.ex. la flexibilité d'accès à l'interface et l'assistance pour alimenter la banque de données par des connaissances personnelles. Ceci a pour conséquence de devoir débattre des stratégies de l'ingénierie du logiciel, afin de pouvoir séparer logiquement l'interface de l'utilisateur de la banque de données.

ZUSAMMENFASSUNG

Für die Entwicklung von wissensbasierten Datenbanken als Hilfsmittel zum Gebäudeentwurf wird der Ansatz, von Benützeranfragen auszugehen, als besonders wirkungsvoll angesehen. Es ergibt sich eine einfache Wissenseingabe und Strukturierung der Datenbank. Die Anfragen geben den Aufbau der Benützerschnittstelle vor und entheben den Ersteller vieler Ueberlegungen. Diese Verbindung führt zu einigen Unterhaltsproblemen. Weitere Gesichtspunkte ergeben sich aus dem Wunsch des Benützers nach Flexibilität im Zugriff und nach Unterstützung bei der Einspeisung eigenen Wissens. Entsprechend müssen Strategien des Software Engineering erörtert werden, damit die Benützerschnittstelle logisch von der Datenbank getrennt werden kann.

1 Introduction

BYGSYS is a knowledge-based system [9] which supports technical building design. The system supports the design of roof structures and it includes textbook knowledge and personal experience knowledge in its knowledge-base. The original aim of BYGSYS was to investigate to what degree experience knowledge could be utilized by an knowledge-based system [1].

Our work comprises a domain analysis and knowledge acquisition and elicitation based on a rather extensive survey of user demands [2]. Furthermore, we performed conceptual modelling, prototyping and testing/evaluation of the BYGSYS system derived from this analysis.

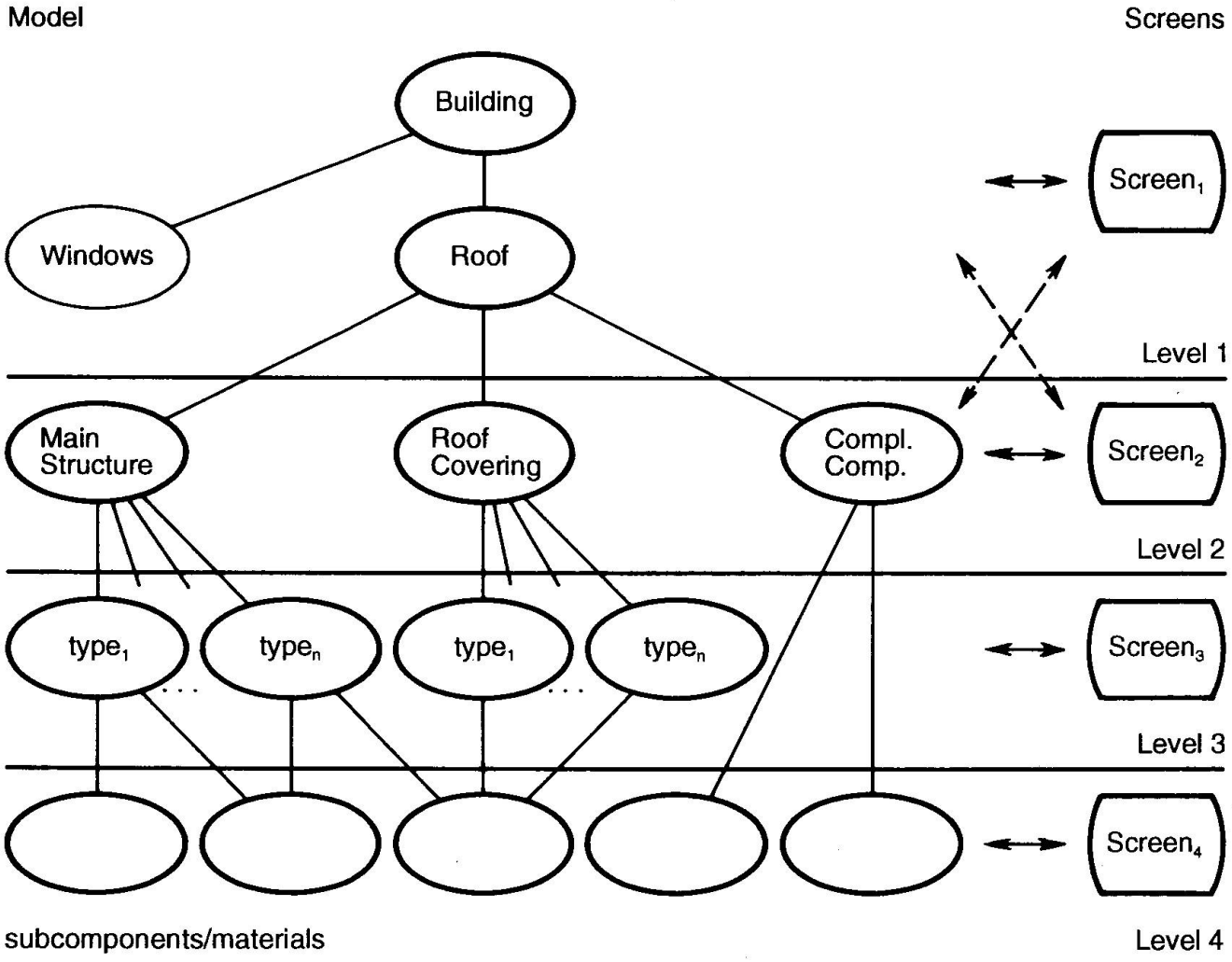


Figure 1. BYGSYS system structure and the corresponding screens.

In the following we will present in a little more detail the method with which we have designed BYGSYS. Thereafter, we evaluate the system functionality and its structure. This leads on to a discussion of software engineering issues that arise due to the development method and due to the impact of user demands.

1.1 Query Driven Knowledge Acquisition and Modelling

An analysis of the nature of technical design knowledge made it clear that the knowledge in this domain refers almost exclusively to physical objects, ie. the components of the building and its roof.

The following acquisition of knowledge where we initially focused on capturing *key queries*, ie. the most important issues for the potential users, emphasized the importance of identifying the components of the roof and their logical relations.

This query driven software design method applied to the BYGSYS system reflects the (simplified) way building design is normally performed. The process can be decomposed into a sequence of tasks, starting with the more abstract issues, ending with the detailed design. The sequence of queries was carefully investigated and approved by a number of professionals, who acted as users. The acquisition resulted in the model of the knowledge-base illustrated in figure 1. The links between queries and the related input/output screens (implementing the user interface) will be presented and discussed in the next section.

This BYGSYS model is equivalent to a product model [3] of the roof extended with some of the roof 'environment' such as the building. The model, shown in a simplified version, forms the core of the system and it can be regarded as a static view of the domain.

Having defined the static structure of the domain through an analysis of user demands - the query driven approach - we then gathered experience knowledge from professionals. This knowledge is represented in the dynamic part of the system and is implemented as procedures and rules on top of the static core structure.

1.2 User Demands

At this point it is important to recognize that user demands affect the system design in two separate ways. Our query driven design approach utilizes the fact that many of the user demands either explicitly or implicitly identify key queries that can direct the design of the knowledge base and the user interface to the system. These demands define *what the user wants to talk about* in the context of the given system.

Secondly, the user demands express *how the user wants to communicate* with this system. These demands of course have a strong influence on the design of the user interface to the system, but as we will see in the following they also have a profound impact on how the entire system should be engineered. The most important of these user demands are listed in table 1.

The system must not constrain the user. The sequence of decisions in a system session must reflect normal practice.
The dialogue should not be tedious and it must be flexible. The designer should be able to skip one or more inputs and possibly changé the default sequence of screens (if this makes sense in the context of the problem on hand).
The designer should be able to augment the system with personal experience.

Table 1. A summary of the most important user demands

As discussed in the above, we found that the sequence of user queries in this domain is (almost) fixed and that it reflects the order of decisions taken in a building design process. We therefore designed input/output screens according to the demands such that each of these correspond to one key query. Thus, the levels in the model and the sequence of screens are strongly connected as the system structure in figure 1 shows.

Please notice that screen 1 and 2 can be chosen randomly, that is, the users can change this particular part of the sequence. This partly satisfies the second user demand of table 1.

The design of the individual screens also supports many of the user demands. The system was designed to avoid too many screens and thus too many questions. A feature is the support for blank entries, so the user only has to concentrate on those aspects relevant to the actual case at the given time of the session. Figure 2 shows one of the major screens illustrating this.

1.3 Analysis

Our knowledge acquisition and modelling strategy has proven very successful. The query driven approach is effective, it speeds up prototyping while retaining the quality of the final system. Furthermore, the queries delimit the domain knowledge and helps focus attention. Using this approach helps avoid the gathering and subsequent implementation of large amounts of irrelevant knowledge. However, we must emphasize that identifying key queries is a time consuming process which has to be performed very carefully.

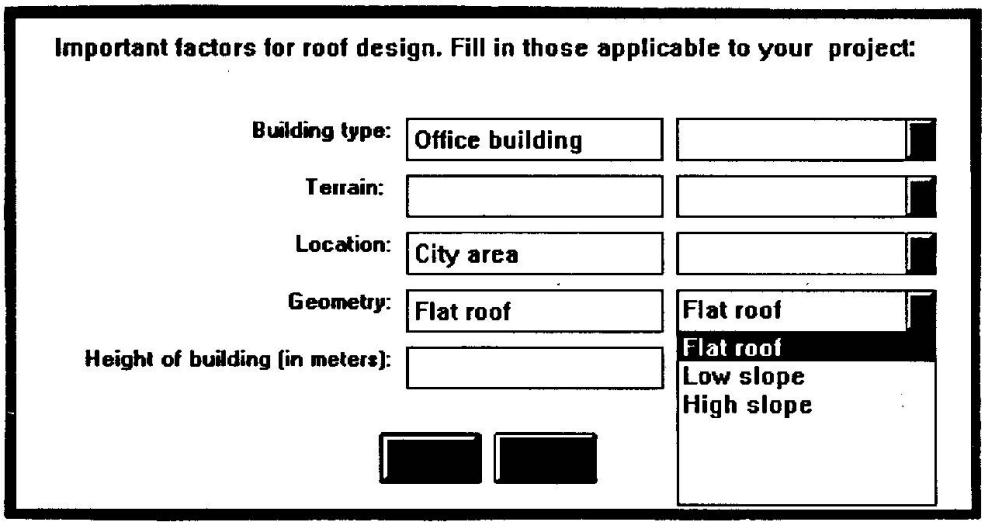


Figure 2. Screen design

The separation between the static model and the associated dynamic experience knowledge supports augmentations of the knowledge-base. More rules can easily be added to the system without revising the entire model. So, user defined knowledge (see table 1) is accommodated.

BYGSYS lets the user augment the system with his/her own rules in a rather simplified way. The rules can be added to the system without any programming skills, but the rules are 'passive' in that they do not interfere or interrupt the actual knowledge processing of the system. They can only support the user during a session, by sending messages and/or warnings to the screen concerning decisions to be made at the actual time. This feature was judged by professionals to be very useful.

In general the BYGSYS user interface was rated as satisfactory by a chosen group of professional users (ie. it met most of their demands). This success naturally lead to investigations into how the knowledge domain could be expanded so that the system could address larger parts of the technical building design process and how some of the more advanced user demands (in particular the possibility for further user defined augmentations of the knowledge base) could be better accommodated.

2 Software Engineering Issues

Some of the more general user demands dealing with flexibility of user interface and with the provision for user defined system augmentations enhance a basic software engineering problem: how to design a flexible and maintainable system. BYGSYS attempts to accommodate these demands to some degree - but not fully. In the following we analyse what impact these demands have (or should have) on the development of design support systems.

The software structure of the existing BYGSYS system does for example not facilitate the introduction of new key queries as discovered by a new analysis of user demands or developments within the field of expertise covered by system. Also, user demands that radically change the sequence of screens can cause system maintenance problems.

2.1 Maintenance of Knowledge-Base

When designing software, the demands of the users are very important, in particular when dealing with interactive decision support systems. The BYGSYS system is heavily influenced by user demands. This is only natural since query analysis was a major factor in the development process. But also user's expectations on screen design and other user interface aspects had an important impact on the system structure.

We have been able to satisfy these demands within BYGSYS. But this is under the condition that the hierarchy of major elements (classes like 'roofing material') is correct and exhaustive and thus that the levels of 'communication' shown in figure 2 are the only ones to address in the domain. That is, in a somewhat narrow domain like roof design the functional core can be defined correctly, once and for all, if this is done carefully.

However, if the knowledge of a domain is more general and/or comprehensive and widespread, a pure query driven approach can be dangerous. It will be very difficult to ensure that all queries are in fact captured before an initial design and system implementation has been completed. It is very likely that unexpected demands will be discovered after this implementation and this might require quite substantial system modifications.

If a new key query is introduced, more functionality of the system is required. The user needs access to a level of knowledge which is not in the current basic structure of the system. In the BYGSYS case we might imagine a future need to include knowledge about a brand new class of roofs with a different breakdown as compared to traditional roof structures. This would cause an extra layer in the model, as visualized in figure 3.

BYGSYS propagate knowledge down the hierarchy in a fashion similar to knowledge propagation in the building design process. Overall decisions are preserved through the stages of design and are strongly connected to detailed decisions. A choice of type of roofing such as tile will for example affect later decisions on many detailed aspects of the roof. BYGSYS reflects this process, and general decisions will have an influence on lower level decisions later in the design session.

One can think of this as knowledge links. High level knowledge, represented in 'high level' objects, are linked to lower level objects. The choice of tile roofing will for example propagate knowledge and information to the "material" objects of the model. These knowledge links are among other things essential in a system that attempts to eliminate trivial and redundant user input by introducing relevant and intelligent options for the users decisions.

BYGSYS has several of these knowledge links which presents a problem when modifying the system by introducing extra layers. All the links have to be maintained when cutting and pasting yet another key layer into the model. Depending on the actual case, it may require a substantial amount of work to introduce this extra key query.

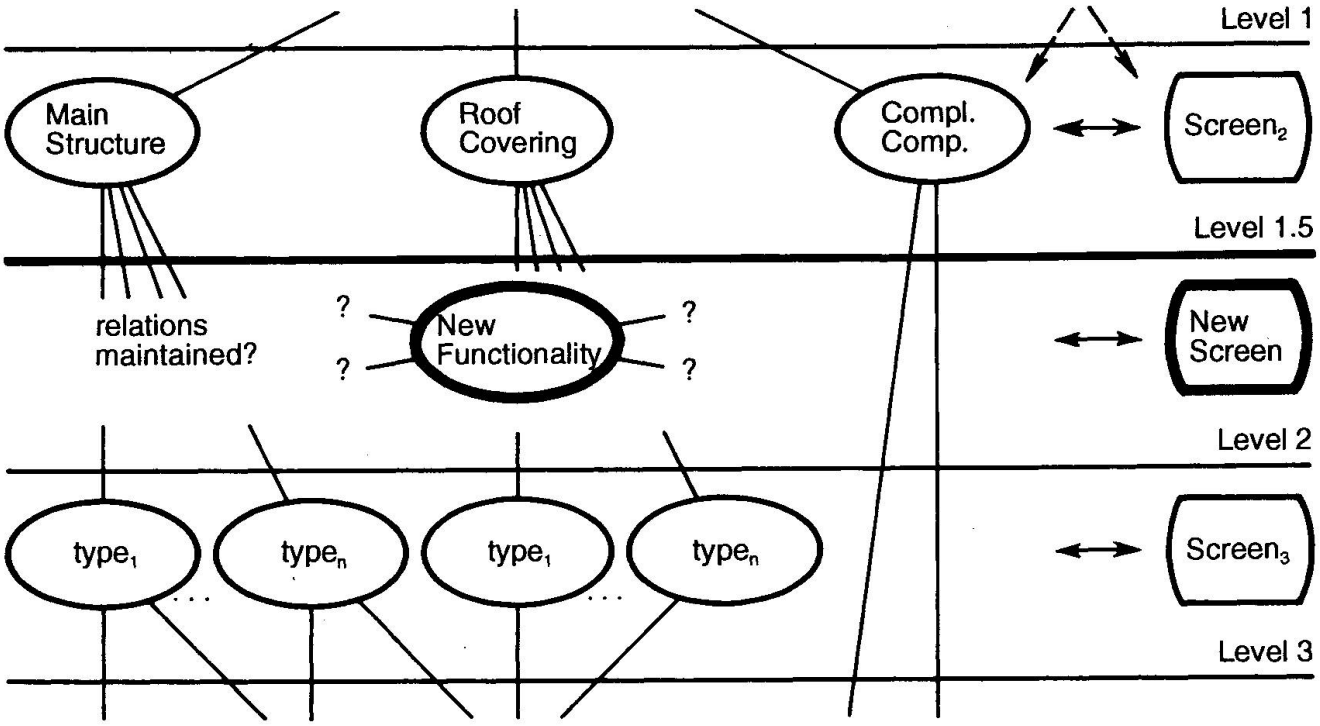


Figure 3. Adding key queries and screens to the system.

2.2 Flexibility of User Interface

If users demand that the dialogue flow, the sequence of screens, be radically changed or be made more flexible, these links again present a problem since all knowledge is propagated top down in the system. If for example we wish to comply with the fact that users may wish to work bottom up, or 'middle out' in certain design sessions we need to change the dynamics of the knowledge base and the screens must be able to deal with the fact that they may be invoked under different circumstances.

So, both the underlying knowledge-base and the associated user interface must be revised in response to these types of user demands. This is complicated by the fact that the user interface is tightly coupled to the structure of the knowledge-base. As illustrated in figure 4, some of the objects in the knowledge-base (UI objects) contain both functionality defined on the given type of knowledge and code dealing with the user interface. Also, calls to the user interface is often a product of the knowledge itself.

2.3 User Driven Augmentations

Both of the above problems have been discussed under the assumption that any changes to the system are the responsibility of a system manager/programmer. However, one general user demand was that user's should be given facilities for defining such changes themselves. It should be possible for users to change the dialogue flow of the user interface and they should be able to augment/modify the underlying knowledge-base. This makes things worse. If the system is not easily maintained by a system programmer then imagine trying to construct an easy to use tool for letting users do the job themselves!

3 Separation of User Interface Software

BYGSYS' problematic lack of separation between its user interface code and its functional core, ie. the static knowledge-base plus rule dynamics, is actually evident in most knowledge-based design support systems. The reason being that software development traditionally focuses on the design of system functionality. The input/output actions defining the system's user interface are seen as (unfortunate) side-effects of computation.

This tight coupling has important advantages such as efficiency of system development and of run-time performance. However, as shown, it does give certain system maintenance problems. These problems are further enhanced by the fact that the development of a user interface for a given system must be an iterative prototyping process due to the diversity of human users and of their work situations.

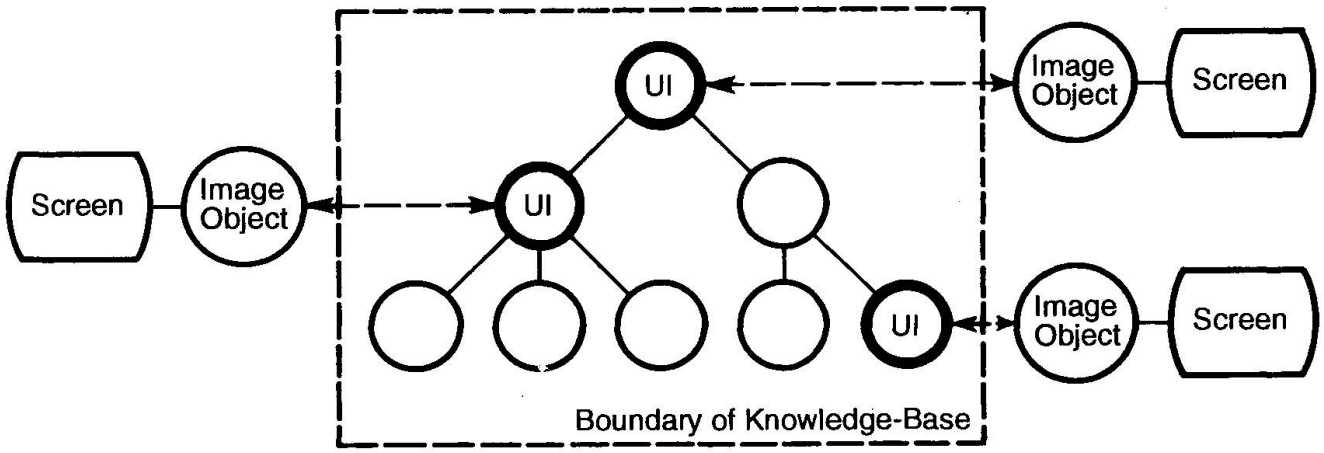


Figure 4. The coupling between knowledge-base and user interface. Within the shell used [7], 'image objects' handle the linkage between screens and knowledge-base objects.

First of all, even minor changes to the flow of dialogue, like the ordering of screens in our BYGSYS interface, will most likely require substantial changes to the entire system. This makes prototyping prohibitively expensive and obstructs maintenance and augmentation in the later stages of the system life cycle. Secondly, the fact that there is no separate explicit and manipulable representation of the user interface (except for the layout and local interaction control handled within the screen controlling image objects) makes it very difficult to construct tools for user defined adaptation and augmentation of the system.

3.1 The Separation Strategy

This motivates the goal of placing the user interface software of an interactive system into a separate module. This module must be used by the functional core for any desired communication with the user. The most important design goal here is to achieve dialogue independence for the functional core of a system so that user interface design modifications do not affect it as long as they do not require additional computations. Also, the user interface software should be isolated from changes to the functional core that do not affect the end users view of the system.

In addition the communication bandwidth required between the two modules should be minimized as much as possible. This will further reduce the interdependencies that compromise maintainability and it will allow the system to be integrated more easily within a networked environment.

The user interface software & technology (UIST) community has been researching this issue intensively for the last 10 years [6]. The main problem with achieving separation is the need for *semantic feedback* [4]. Feedback on the semantic effects of user inputs requires the user interface to have knowledge of or access to the state of the functional core. In more advanced graphical user interfaces where the user can manipulate data directly and where feedback on this must be given continuously, this becomes a serious problem.

In the more traditional query based interfaces that are most applicable to the design of a design support system like BYGSYS this is fortunately not a real problem. Here it is sufficient to exchange semantic information between the user interface and the functional core at closure points in the dialogue. This means that it should be possible to come up with a sensible development strategy for such systems!

The strategy for separating user interface from functional core must deal with 1) how the inevitable exchanges of information across the separation interface are to be coordinated and 2) how information is distributed between the two modules so as to minimize this communication. These two aspects are covered by the *global control model* and the *information distribution model.*

3.2 The Global Control Model

The *global* control mechanism governs the flow of control between the two domains whereas *local* control mechanisms handle dialogue flow within the user interface domain and the algorithmic flow within and across functions of the functional core. The global control model must define which component is in charge of the information exchange and whether this control may shift between components. Three distinct strategies have been proposed [4]: the *computation dominant control*, the *dialogue dominant control* and the *mixed control* models.

Systems like BYGSYS that employ the computation dominant control model let the user interface software be controlled by the functional core. The user interface is implemented by a set of abstract interaction devices controlled by the functional core. The *image objects* used in the BYGSYS implementation for controlling the screens exemplify such abstract interaction devices.

The computation dominant model has two basic problems. First, the *overall dialogue sequencing* is only implicitly defined by the local control flow of the functional core. This compromises dialogue independence and consistency and it furthermore discourages modifications of a given user interface design to a functional core.

Second, user interfaces constructed with systems employing this separation strategy tends to exhibit a *highly moded structure*. The functional core decides when the user may input a certain piece of information according to its internal control flow, eg. when the problem solving mechanism needs new data to continue processing the knowledge base. Since its execution is temporarily stopped until this information is returned a moded structure is forced onto the dialogue.

The dialogue dominant control model is a better approach to system design. Here the functional core is accessed by a set of functions controlled strictly by the user interface module. This avoids the above problems but introduces new ones. These functions must be atomic. During their execution they may not communicate with the user, eg. to retrieve missing data or to communicate error conditions arising within the functional core.

The mixed control model is therefore (in principle) the best solution. This model extends the dialogue dominant model by allowing functions in the functional core to call the user interface during execution. This is the model assumed by most contemporary user interface toolkits such as the X toolkit [10].

3.3 The Information Distribution Models

The second and generally most tricky problem in defining the separation interface is to specify the distribution of data across the interface so as to minimize communication bandwidth while still achieving dialogue independence.

This *information distribution model* must specify the kinds of information residing in each module, how and at what level of abstraction the two domains exchange information. It must define how output objects from the functional core are handled within the user interface and must decide whether or not user interface mechanisms that handle semantic feedback have direct access to the internal data of the functional core.

As briefly discussed in the above, this model is not critical in the design of a query/answer system like BYGSYS where information only needs to be exchanged in relatively small units (slot values in knowledge objects) and at clearly identified closure points (each time a level in the hierarchical knowledge structure is processed). Nevertheless, the approach taken will have an effect on the maintainability of the system.

BYGSYS employs the *shared data* information distribution model. All information within the functional core that is relevant to the user interface is placed in a shared data structure. In casu BYGSYS this is actually the knowledge base itself, the image objects can retrieve slot values in the knowledge base directly (eg. for displaying legal choices in a screen). Also, they have direct access to the rules and procedures triggered by the various choices made by the user within the screen.

This approach of course facilitates the user interface management of semantic feedback and it relieves the functional core from considering user interface aspects such as text formatting etc. However, the model obstructs system maintenance - all the direct references must be maintained.

The *distributed data* information distribution model seems to be a better approach. It tries to address these problems by suggesting that the internal data be distributed. The functional core defines a relevant view of its internal database through a set of mapping functions. Updates in the view are explicitly communicated between user interface and functional core. Low bandwidth across the separation interface can be achieved so this is an ideal model.

4 Guidelines & Conclusions

This discussion of separation strategy leads to some guidelines for constructing maintainable, flexible and user augmentable design support systems. Traditionally, a knowledge acquisition phase is followed by a conceptual modelling phase whereupon the system can be constructed.

4.1 Guideline

The query driven approach provides a good starting point for performing domain analysis and knowledge acquisition. It may also be put to good use in the context of the actual system design. However, it is at this point that we should take the above separation issues into account.

Instead of focusing solely on the construction of the knowledge-base and postponing most user interface considerations, these should be developed more or less in parallel. This is actually supported by many contemporary knowledge-base shells where eg. image objects and screens can be defined and used within the scope of the shell.

However, the shells do not directly support the construction of systems with a clean separation between user interface and functional core. The overall flow of dialogue and the presentation of data must still be handled within the functional core.

We recommend the introduction of a separate *linkage module* [5] between the knowledge-base and the user interface functionality. This module must contain an explicit and manipulable representation of the dialogue, it must implement the chosen global control model and information distribution model.

This linkage module may just be a conceptual entity - representing the fact that the designer does strive for separation - or it may be realized as a physical entity. Intelligent user interfaces [8] may employ user modelling techniques in order to adapt to users. Others may embody handwriting/voice recognition and/or natural language understanding. In these contexts a *blackboard architecture* with three separate knowledge-bases could be a sensible implementation strategy.

4.2 Conclusions

One important conclusion of the above is that not only does the user demands impact knowledge-base design and user interface design but they also impact the approach to software design that must be taken when designing the given design support system.

User (query) driven modelling and design does without doubt imply incremental prototyping. However, in the case of larger systems and/or complex knowledge domains, this is only feasible if care is taken to have optimal separation between the user interface and the functional core. An initial prototype can easily be developed without separation (as supported by most shells) but later revisions of this design will most likely be prohibitively complex.

So, user demands must be taken seriously when designing decision support systems. We recommend user (query) driven knowledge acquisition and modelling but combined with an overall strategy for software separation.

5 References

1. T. Andersen; A. Gaarslev, Building faults - preventing future faults by utilizing past experience, in: V.Ireland (ed), Proceedings of CIB90 vol.7, 1990.

2. T. Andersen, BYGSYS technical report, Dept. of Construction Management, June 1992.

3. B.-C. Björk, Basic structure of a proposed building product model, CAD-journal, vol 21(2), March 1989.

4. N.V. Carlsen, Towards a Common Context for User Interface Management System Design, in: C.Unger & J.A.Larson (eds) Engineering for Human-Computer Interaction, to be published by Elsevier Science Publishers, 1993.

5. G. Cockton, A New Model for Separable Interactive Systems, Proceedings of INTERACT'87, Elsevier Science Publishers, 1987.

6. H.R. Hartson; D. Hix, Human-Computer Interface Development: Concepts and Systems for its Management, ACM Computing Surveys, volume 21(1), March 1989.

7. Intellicorp, User's Manual, KAPPA-PC, 1991.

8. J.W. Sullivan; S.W. Tyler (eds), Intelligent User Interfaces, ACM Press, Addison-Wesley, 1991.

9. D.A. Waterman, A Guide to Expert Systems, Addison-Wesley, 1986.

10. D.A. Young, The X Window System - Programming and Applications with Xt (OSF/MOTIF Edition), Prentice-Hall, 1990.