A Knowledge-Based Support System for Embedded Computer Software Analysis

Kari Hakkarainen, Tuomas Ihme & Markku Oivo

Technical Research Centre of Finland (VTT)
Computer Technology Laboratory
P.O. Box 201
SF-90571 Oulu
FINLAND

ABSTRACT

The system specification support environment presented in this paper, **PROSPEX** (Product Specification Expert), supports the RT-SA methodology. It not only provides graphic support for drawing, but also helps the engineer in the analysis and design process itself. PROSPEX analyses the quality and correctness of the RT-SA diagrams using knowledge-based techniques. PROSPEX also includes features for supporting design reuse. The development environment of PROSPEX was KEE expert system development shell running on a Symbolics workstation[1].

Keywords: Software Engineering, Structured Analysis, Knowledge-Based Systems, Reuse

[1] This research was carried out as a part of the FINPRIT research programme, funded mainly by the Technology Development Centre of Finland (TEKES). Financial support was also provided by the Technical Research Centre of Finland (VTT), Kone Corporation, Nokia-Mobira, and Edacom.

1. Introduction

The specification and design of real-time embedded systems is a complicated and error prone task. A good design methodology is an essential prerequisite for the successful completion of the activities. Yourdon's RT-SA/SD (Real-Time Structured Analysis and Design) is among the most widespread methods developed to assist in the initial phases of developing embedded real-time systems.

The origin of RT-SA/SD stems from the structured design techniques developed during the 1970's. The concept of data flow techniques, an essential part of RT-SA/SD, was developed by Tom DeMarco [1979]. He integrated data flow technique with the earlier principles of structured design [Yourdon & Constantine 1979]. Another important feature offered by the methodology is the view to the data content of the system by using entity-relationship representations. Ward & Mellor [1985] augmented these methods by introducing tools for representing the dynamic behaviour, that is control and timing, which is essential in real-time systems. One of the reasons for the popularity of the RT-SA/SD methodology is that it integrates the three above mentioned representations in a systematic and uniform way.

Several computer aided tools have been introduced to support the RT-SA/SD method [CASE Outlook 1988]. The recent tools have acceptable graphics capabilities for drawing the diagrams, but most of the commercial tools are scarcely more than drawing assistants. They don't have profound knowledge of the design methodology, not to mention any support for reuse.

The prototype presented in this paper, PROSPEX (Product Specification Expert), supports the real-time structured analysis (RT-SA) methodology. It differs from the commercially available tools in that it not only provides graphic support for drawing, but also helps the engineer in the analysis and design process itself. PROSPEX analyses the quality and correctness of the RT-SA diagrams using knowledge-based techniques.

PROSPEX also includes features for supporting design reuse. In this, the goal of PROSPEX is to provide tools for developing reusable designs, and using a collection of them in industrial production of embedded computer systems.

The PROSPEX prototype[1] was built in a three-year research project that studied the state-of-the-art of real-time structured analysis support. The goal of the project was to study and refine methods and tools that meet the needs of real life analysis better than the current commercial tools. The ideas developed within the project were demonstrated and highlighted through a prototype that realized the most central features of the mentioned support. The actual implementation is discussed in more detail in [Hakkarainen, Ihme & Metcalfe 1989] and in [Metcalfe, Hakkarainen & Ihme 1989]. Reuse is discussed further in [Ihme 1989].

---

[1] In fact, we have two different prototypes. The features between the two prototypes will not, however, be distinguished here.

The knowledge-based PROSPEX prototype was developed in KEE expert system development shell running on a Symbolics workstation. We applied knowledge-based techniques primarily for two different reasons. First, one of the objectives of the project was to study how well knowledge-based techniques fit into implementing this kind of tools. We did not want just to make use of already existing resolutions, but also wanted to have a look into the future possibilities (Figure 1). The other reason for using knowledge-based tools was that it would have been very difficult to implement the wanted facilities with traditional approaches. Advanced knowledge-based techniques provide the possibility of demonstrating and testing ideas in early phases, which is not the case when using traditional tools.

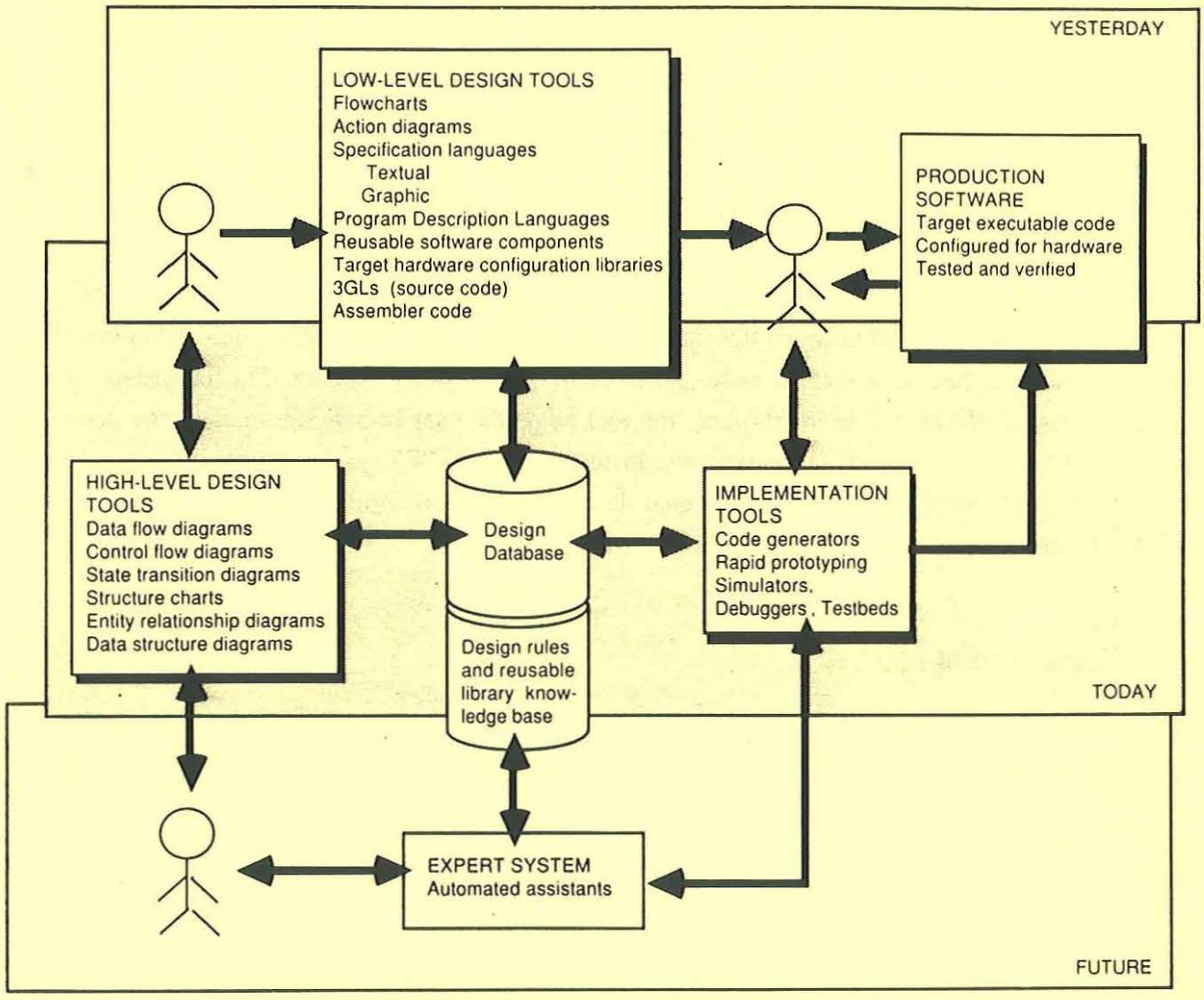

Figure 1. Code generation yesterday, today and tomorrow. [CASE Outlook 1:6, p. 18. Reprinted with permission of CASE Consulting Group, Portland, Oregon, USA].

2. System Overview

PROSPEX can be used in two different configurations. The basic PROSPEX, i.e. the kernel, includes support for the RT-SA method. This configuration is totally application independent. The larger configuration (C-PROSPEX) includes support for the creation of reusable designs, as well as application specific domain knowledge. The two configurations appear to the user as one integrated tool, and the partitioning is more a matter of implementation.

2.1 User Perspective

PROSPEX appears to the user as an intelligent RT-SA assistant. It offers the user the basic environment with graphical and textual tools for creating and maintaining RT-SA documents. The user interface, i.e. the external notation, conforms the RT-SA syntax, while the internal representation is geared towards efficiency and flexibility. The internal structure and the rationale behind it are further discussed in [Metcalfe, Hakkarainen, & Ihme 1989].

PROSPEX constantly controls the quality and correctness of diagrams during creation and modification as a background process. Because an incomplete diagram cannot be analyzed thoroughly, additional analyses can be activated once the system models are complete. The emphasis of the implementation of PROSPEX is in this area: the tool helps the user in judging whether the design is correct and if it does meet quality requirements. In addition PROSPEX guides different users to create uniform design documents. The ultimate goal is to create more understandable documents and facilitate easier maintenance.

2.2 Implementation

The logical structure of the PROSPEX prototype is layered (Figure 2). The basis for all the different tools incorporated in the basic PROSPEX is the intelligent model editor. It is a totally application independent package of graphical and textual tools for creating and editing RT-SA diagrams. The kernel also includes a RT-SA primitive knowledge-base and a methodology rule base (Figure 3).

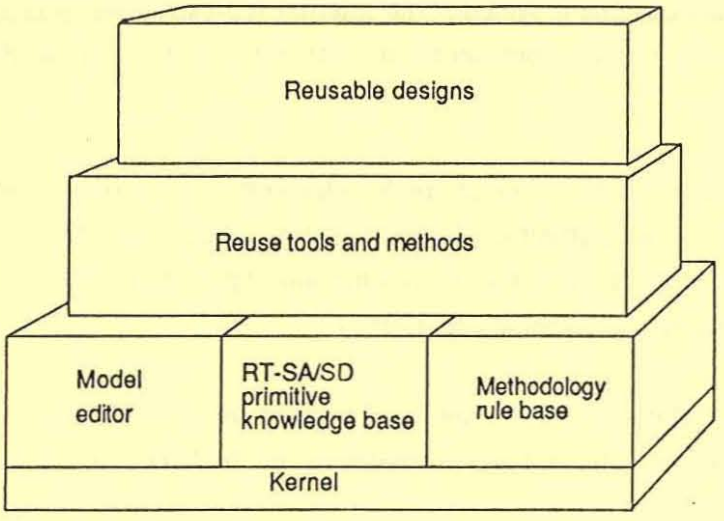

Figure 2. The layered architecture of the PROSPEX-prototype.

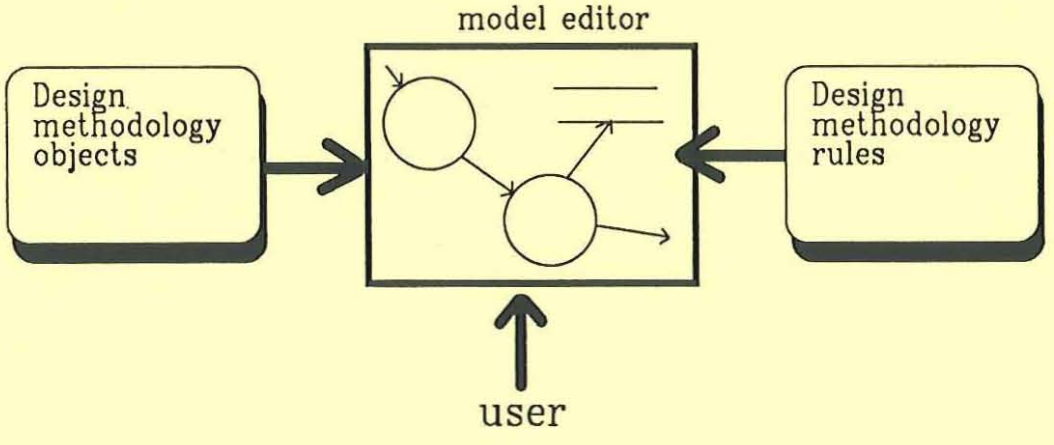

Figure 3. The main components of the PROSPEX kernel are a graphical model editor, a design primitive knowledge base and a design methodology rule base.

The intelligent model editor with related menus serve as the overall user interface for creating, viewing and modifying RT-SA diagrams. The design primitive knowledge-base has definitions for all the RT-SA primitives and for their graphical appearance. The primitives are used to create internal frame based models of diagrams as the engineer designs a system. The internal representation was geared towards efficient manipulation, whereas the external representation conforms totally with the RT-SA notation.

Each diagram and its elements are represented by a set of model elements having one or several graphical instances. All the user interface actions affect the internal system model, and the graphical diagrams merely mirror the model. Having the graphical representation separate from the system model makes modifications easy and helps in guaranteeing consistency.

The user interface incorporated in the model editor is used, not only for drawing the diagrams, but also for activating various analyzing functions. The related knowledge is declared in the design methodology rule base.

3. Design Support

The user interacts with the PROSPEX prototype via a set of graphical and textual tools (Figure 4). Through the graphical model editor, icons and menus the user inserts, modifies and evaluates designs, or system models.

3.1 System Modelling

The creation of a model can start from any one of the context, state transition, or entity relationship diagrams, or the events list. Normally, the context diagram or events list would be chosen, but PROSPEX supports any method. The user opens windows ("viewports", in the KEEpicture terminology) and chooses elements from a menu which is accessed via the mouse. Additional diagrams are opened as necessary. Editing is also mouse-and-menu based.

The RT-SA documents are constantly analyzed in the background to reveal errors in consistency and completeness during drawing. The constant background control ensures that all the prerequisites needed in using the method successfully are met. The rules ensure certain features, such as completeness and uniqueness, and reveal unwanted ones, such as logical malfunctions.

The modelled system can be both analyzed and viewed at any point. Various types of checks can be initiated from the menu; in addition, certain illegal constructions (for example, having a input flow to a transformation with an illegal source in the context diagram) are highlighted immediately.

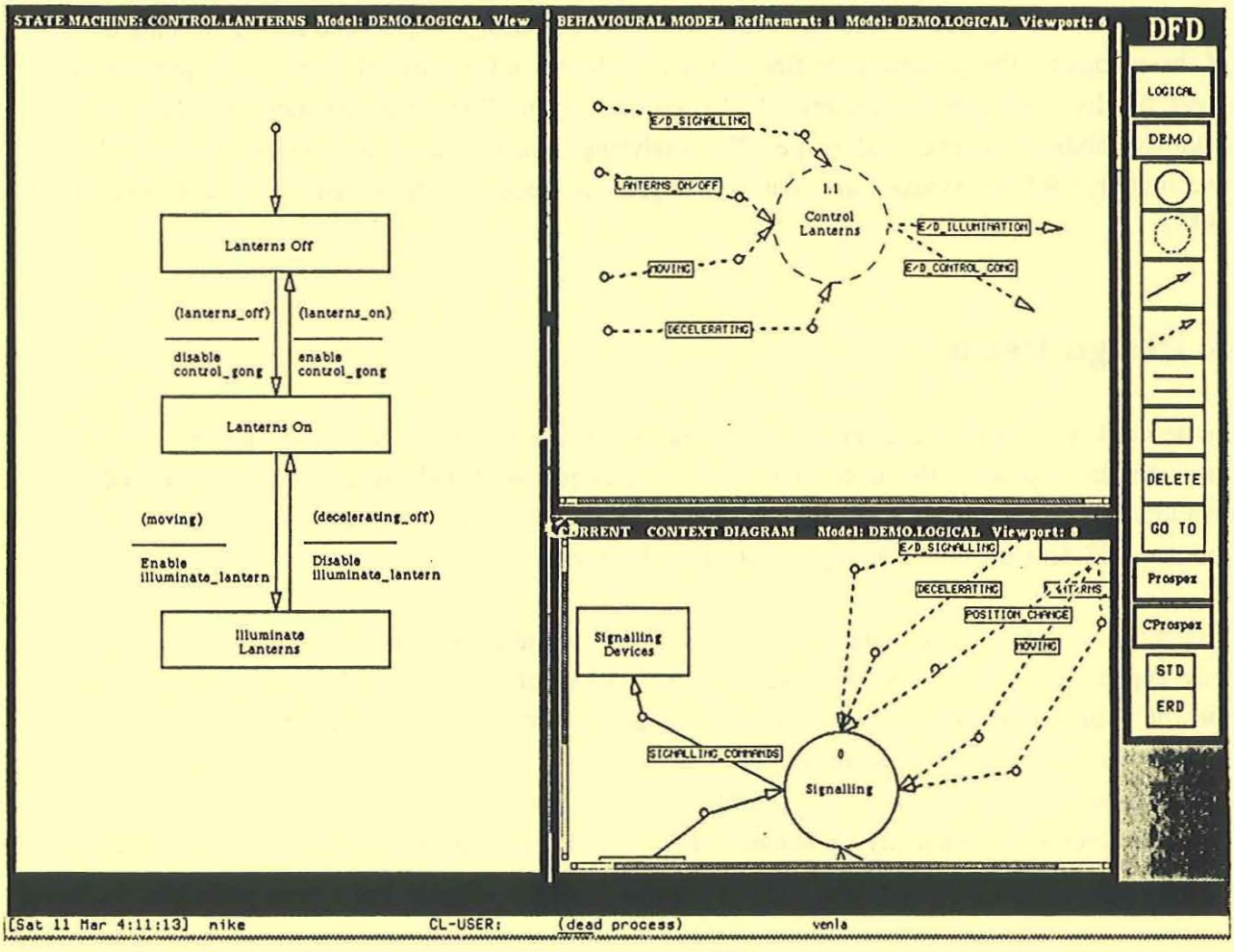

Figure 4. The user interfaces with PROSPEX through an intelligent model editor. In addition to manipulating RT-SA models, the editor is used to start various menu-based analysis functions.

In addition to checks and tests of correctness, a PROSPEX user can access information on the logical structure of the created design. For example, a hierarchy of transformations can be displayed from the top-level transformation.

A session can be terminated at any point, and the knowledge bases containing the modelled system will be saved. Similarly, a system can be recalled for either editing or checking at a later date [Oivo & Hakkarainen 1988].

3.2 Model Verification

Only partial design analyses are meaningful during the design process. More complete analyses can be activated either automatically or by the user once a goal or subgoal of the design process is reached. In the rule base, the design methodology is defined in terms of design objects and rules governing the use of these objects. The purpose is to find out how well the RT-SA model meets the requirements of good quality and correct features of the product itself. The characteristics analyzed include communicability, naming, and scope. The analyzing function does not deal so much with the unconditional RT-SA syntax rules, but with aspects of good quality design [Oivo & Hakkarainen 1988].

4. Design Reuse

Reuse has been recognized as one of the key factors in trying to improve quality and productivity in software development. However, most of the attention in CASE is directed toward tools for developing new software [CASE Outlook 1988], and there is still quite a gap between reusability ideas and their efficient and effective implementations [Seppänen 1987].

Domain-independent reuse systems can be made flexible and generic, but it is commonly known that only application specific systems reveal the real power of reuse. PROSPEX provides a domain-specific infrastructure that relies on a quite wide spectrum of the RT-SA domain-independent system design method. PROSPEX is an approach to explore and demonstrate how to develop reusable designs, and how to use a collection of them in industrial production of embedded computer systems. These problems are mutually dependent. The former presupposes knowledge of how reusable resources are going to be reused, and the later the existence of appropriate reusable resources.

4.1 Reusable Knowledge

Production of reusable components is based on the domain structure and models [Prieto-Diaz 1987]. The approach is similar to that used in object oriented development. Some of the features desirable for components supporting software reusability according to [Goguen 1984], [McCain 1985], [McCain 1986] and [St. Dennis et al. 1986] are taken into account in PROSPEX.

The interface of a component can be considered as a socket into which users can plug devices. Each facet of the socket must be clearly defined [St. Dennis et al. 1986]. The syntax must not only be correct, but should also convey some meaning. To achieve composability and plug compatibility, actions based on results of the component should be made at the next level up. The type of parameters should be defined (i.e. typed), and the parameters should be checked and constrained to satisfy the domain of the problem to be solved.

The components of PROSPEX are RT-SA basic items or composite components, whose types are the characteristics of a product. A composite component is created from the RT-SA basic items and from other composite components. The typed components are saved in a component library.

A reuse support system incorporates a variety of different knowledge types, each of which may require representation and retrieval mechanisms of its own. Examples of different knowledge types are:

- construction knowledge,
- inspection knowledge,
- design knowledge,
- production process and target environment knowledge,
- functional knowledge of components,
- design analysis, and
- product consultations.

The most visible of the knowledge types, the design knowledge, decomposes a complex design problem into simpler design steps and guides the user through these simple design steps toward the complex design goal. Design knowledge builds design kits, which provide prototype solutions and examples that can be modified and extended to achieve a new goal instead of starting from scratch. The design kits use component, construction and inspection knowledge to monitor, check and control the design process. In the early design phases most of the design and reuse decisions are made on the product model and subsystem levels.

4.2 Reuse Based Design Process

The model building process of the RT-SA method proceeds by deriving features of the model from previous models and documents. We have built a trace mechanism to document this derivation process. This documentation plays an essential role for reuse afterwards when modifying or maintaining the model or reusing the model in building other models of the current system or models of other systems.

A snapshot situation in a reuse based software design process is shown in Figure 5. At the top of the figure there is the library of reusable components. Systems are completely modelled RT-SA systems and consist of components from the lower levels. Subsystems are composite components modelled up to the terminator level of the RT-SA method. Composite components are combined from basic components and other composite components. The basic components are typed RT-SA items.

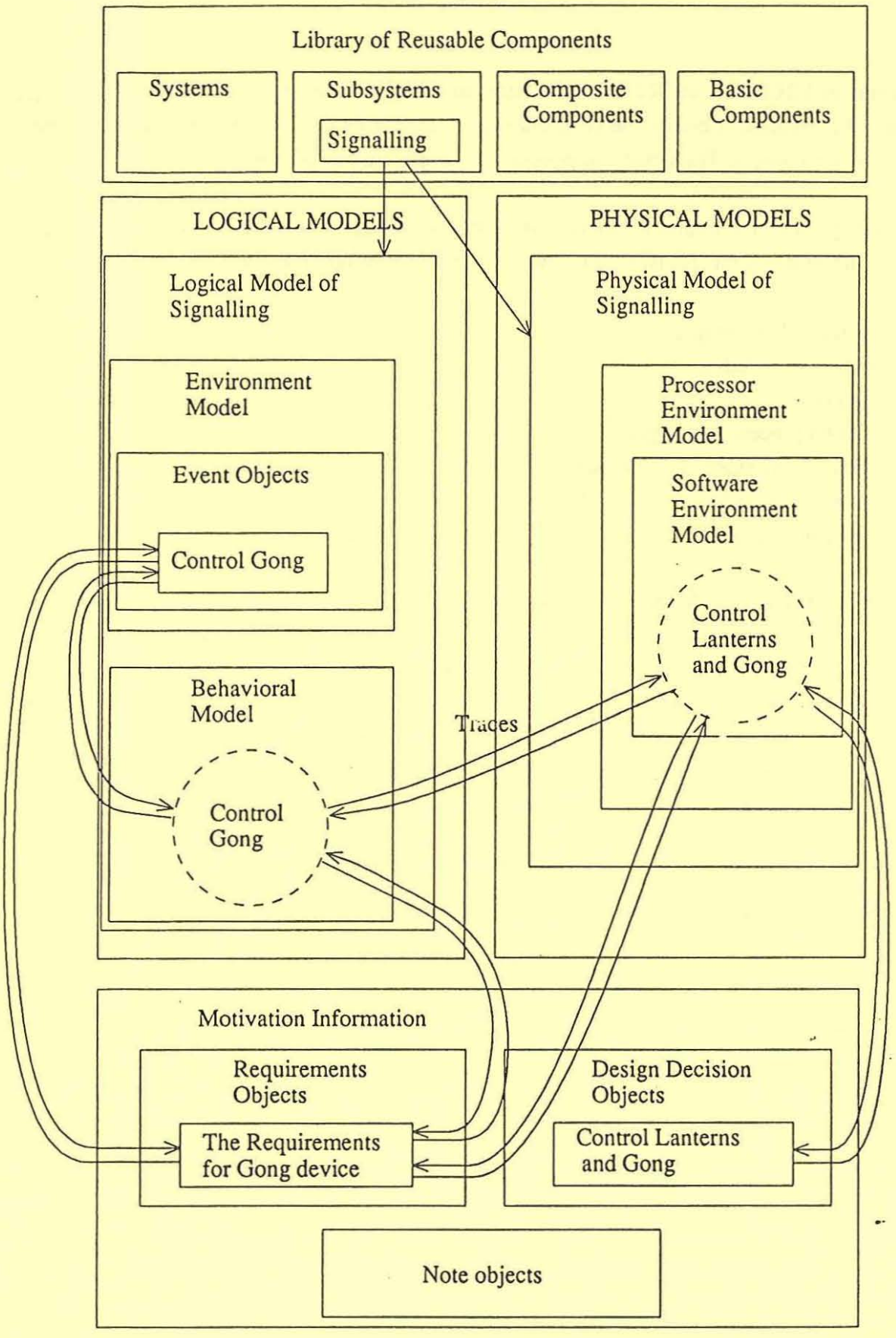

Figure 5. An example of reuse based software design process. The basic reuse mechanisms can be seen as an integral part of the RT-SA methodology, and the reuse component structure is similar to that of normal RT-SA components. In addition to usual RT-SA support, PROSPEX provides also support for automatic tracing, storing of motivation information, and use of library components.

The logical model building process proceeds by deriving features of the logical model from narrative requirements and previous features of the model [Ward & Mellor 1985] and by reusing the logical models of components in the reuse library. The elements of the environment model are external events, terminators, flows, and stores on the context schema. Because the most likely omission in a narrative requirements document is a systematic description of event dependencies[Ward & Mellor 1985], we have implemented event objects that describe an event and define its inputs and outputs.

The physical model is derived from the logical model and from previous features of the physical model under the constraints of the narrative requirements document, and by reusing the physical models in the reuse library (Figure 5). The trace objects are used to describe and check the connections between the logical and physical models.

Motivation objects are used to capture and describe useful background knowledge of components. Individual requirements in the narrative requirements document are saved in requirements objects, that are used to construct traces between the narrative requirements and the models. A Design-Decision object includes the recorded design decisions about the object. Note objects gather comments and temporary motivations that cannot be assigned to any above information type when they are created.

5. Experiences and Future Research

The ideas presented in this paper were developed within a three-year research project. Central issues have been demonstrated by the PROSPEX prototype that was built using the KEE tool on a Symbolics workstation. The prototype implemented the basic mechanisms described in this paper for the creation, manipulation and analysis of designs based on Yourdon's RT-SA methodology. The prototype is not a complete product, but was rather built to highlight the research results. The features of the prototype should thus be seen in that respect. Further discussion of the implementation can be found in [Metcalfe, Hakkarainen & Ihme 1989].

Knowledge-based techniques proved well-suited in demonstrating and evaluating the results of a research project. The knowledge-based tools were found befitting when prototyping the implementation of ideas. The software and hardware environment we used is, however, too complicated, inefficient and clumsy for implementing a full scale structured analysis tool.

As follow-up we have initiated the Reuse Assistant project which studies and develops knowledge-based support for reusing structured analysis designs in industrial embedded computer software production. The key issues in that project are:

1. To develop methods for building, using and maintaining libraries of reusable components that can be applied to practical software construction in the short term.

2. To integrate existing approaches that support practical reusability in the construction of embedded software systems.

3. To evaluate and demonstrate the application of methods and representations in the construction of a library of components in an embedded system domain, and demonstrate its use in the practical construction of software systems.

6. Conclusions

Using knowledge based techniques makes it possible to implement extensive analysis of the correctness and quality of RT-SA documents. It can provide an engineer with a sophisticated RT-SA tool instead of mere drawing support. This enables the engineer to concentrate on the creative and most fundamental tasks of a design process.

Knowledge representation mechanisms are very important from the viewpoint of system functionality. By having an efficient internal representation and by separating the models from their graphical appearances we guaranteed efficiency, extendability and easy modifiability. Knowledge-based techniques proved to fit well in prototyping a tool such as described in this paper, but use of them in implementing a real scale tool should be evaluated carefully.

Domain-independent reuse requires complex techniques for acquiring reusable components, and strongly domain-specific approaches need sophisticated methods for capturing the domain knowledge. PROSPEX is a knowledge-based software design reuse system that combines the benefits of the two approaches. It exploits application specific knowledge by following established reuse practices.

PROSPEX supports both the development of reusable designs and the use of them in the industrial production of embedded computer systems. For development, PROSPEX provides a graphics-based user interface and an effective frame-based knowledge representation scheme. A menu-based library component catalogue and graphical visualization facilities are also available.

Once the components are properly specified, it is possible to reuse the same component in different product families and in different versions of one product. Product knowledge can also be used in semi-automating some design steps and giving intelligent advice and product consultations. It moves a considerable amount of the analysis and design effort of embedded real-time systems into defining the product itself.

References

CASE Outlook 1988. CASE Tools for Reverse Engineering. CASE Outlook 2, 2, p. 1.

DeMarco, T., 1979. Structured Analysis and System Specification. New York, Yourdon, Inc. 352 p.

Goguen, J. A. 1984. Parameterized Programming. IEEE Transactions on Software Engineering SE-10, 5, pp. 528 - 543.

Hakkarainen, K., Ihme, T. & Metcalfe, M. 1989. PROSPEX: A Knowledge-Based CASE Tool. The Second Scandinavian Conference on Artificial Intelligence. June 13-15, 1989, Tampere, Finland.

Ihme, T. 1989. A Knowledge-Based Support System for the Reuse of Structured Specifications and Designs of Embedded Computer Systems. The First Nordic Conference on Advanced Systems Engineering May 9-11, Kista, Sweden.

Ihme, T., Hakkarainen, K.& Kurki, M. 1988. Knowledge-based Support for Software Design Reuse, Finnish Artificial Intelligence Symposium, STEP-88, Helsinki, Finland, August 15 -18 1988.

McCain, R. 1985. A Software Development Methodology for Reusable Components. Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences 1985, pp. 319 - 324.

McCain, R. 1986. Reusable Software Component Construction: A Product-Oriented Paradigm. (unpublished) IBM Federal Systems Division, Houston TX.

Metcalfe, M., Hakkarainen, K. & Ihme, T. 1989. A Structured Analysis and Design Tool Based on Frames and Relationship Modelling (in preparation).

Oivo, M. & Hakkarainen, K. 1988. A Knowledge-based Support System for Embedded Computer Software Analysis and Design. Finnish Artificial Intelligence Symposium, STEP-88, Helsinki, Finland, August 15 -18 1988.

Prieto-Diaz, R. 1987. Domain Analysis for Reusability. Proceedings, Compsac87, Tokyo, October, pp. 23-29.

Seppänen, V. 1987. Reusability in Software Engineering. In: Freeman, P., Tutorial: Software Reusability, IEEE Computer Society Press, Washington D.C., 1987, pp. 286-297.

St. Dennis, R. et al. 1986. Measurable Characteristics of Reusable Ada Software. Ada Letters 5, 2, pp. 41-49.

Ward, P. & Mellor, S.J., 1985 Structured development for Real-time Systems, Vol 1...3, New York, Yourdon, Inc.

Yourdon & Constantine 1979. Structured Design, Prentice-Hall, Englewood-Cliffs, New Jersey.