A Knowledge-Based Simulation Environment for Electronics Design

Timo Ketonen and Jukka K. Nurminen
Nokia Research Center
Espoo
Finland

ABSTRACT

In this paper we discuss an object-oriented simulation environment that couples symbolic and numeric computing. Symbolic computing is mainly used to facilitate the use of complicated numeric design tools, such as simulators and optimization algorithms. Instead of studying new techniques for design analysis or synthesis, the goal of our research has been to develop a framework which enables easy and efficient use of existing tools. Object-oriented techniques are used to provide the necessary flexibility. Application and tool specific knowledge, together with a graphical user-interface, take care of design routines and assist users in operating different design tools. The techniques have been applied in a design system for mobile telephones which is currently in use in a number of design groups at Nokia Corporation. Various aspects of this on-going work have already been discussed in e.g. [1-4].

INTRODUCTION

Graphical user interfaces have significantly simplified the use of complex mathematical tools. However, the user still has to take care of tasks like selecting proper tools for a design task, formulating models for the tools and defining proper parameter values for the efficient and reliable operation of the tools. In these tasks he needs guidance and counseling especially if he is not a frequent user. The required expertise is difficult to learn since many tasks, like design, require the use of a number of different tools most of which are used casually during certain design phases.

The concept of coupling numeric and symbolic computing (for an overview see [5]) shows promise in helping with the above problems. Symbolic computing can be used to assist in the selection and use of numeric tools and in the interpretation of results.

Object-oriented techniques have attracted considerable attention both in the simulation community and as a useful technique to develop modular and flexible graphical user interfaces. To allow for a modular, easily expandable framework to be created, object-orientation is seen as an important component.

In this paper we describe how the above techniques have been used in RFT (an acronym for Radio Frequency Tools) design environment to facilitate the use of different design tools.

BACKGROUND AND SOME REQUIREMENTS

The application field, radio frequency design, is characterized by a small number of components with highly complex interactions. Mathematical modelling of the interactions is not easy and different tools and algorithms are needed to solve the resulting numerically difficult problems. Since design is typically an iterative process analysis tools are repeatedly used in the evaluation of the performance of the designs. This increases the need to perform the analysis easily, and preferable automatically, to save design time.

Another feature of importance is the hierarchical organization of the design process. First, an initial functional level design is developed which on a crude level fulfills the specification. Parameters selected at functional level form the specification of the component level design. And once the component level design of a functional level block is completed the performance measures are propagated back to functional level to see how the actual component level implementation affects the overall behavior of the device.

In order to be useful in practical design work the design environment should have

* one graphical interface for all integrated tools

* the ability to use existing design tools and allow easy integration of new ones

* flexible communication which allows different design tools to co-operate

* support for hierarchical design

Existing design environments do not fulfill these requirements very well. Although graphical interfaces are getting more and more popular, most interfaces are developed to take care of the use of only one tool. Very few tools allow the users to integrate existing design tools and design conventions in a simulation environment. From the practical point of view this is an essential requirement since in order to keep up with the latest technology the system must allow the addition of new tools and algorithms. When all tools are hidden behind a common user interface, changes in the integrated tools do not cause any additional difficulties to the user.

RFT DESIGN ENVIRONMENT

The main motivation in the development of RFT has been the creation of an environment that is useful in practical design work. The application of object-oriented architecture and symbolic computing has made it possible for RFT to fulfill the above requirement fairly well.

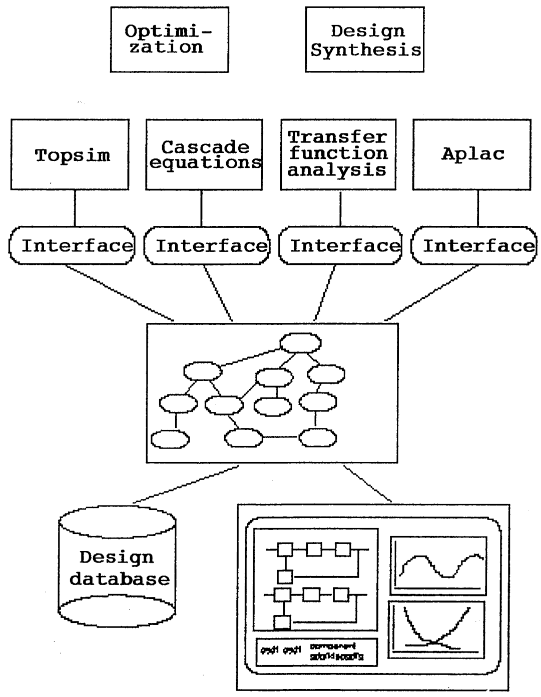

Figure 1. The general architecture of RFT

Fig. 1 represents an overview of the architecture of RFT. Design entities are implemented using object-oriented programming. Interfaces to the basic design tools manipulate this representation. The basic design tools include simulators, like TOPSIM [6] and APLAC [7], and application specific analysis tools. More advanced tools for simulated measurements, optimization and rule-based design synthesis control the use of basic tools and perform operations that require co-operation. The graphical user interface allows users to create and modify design diagrams in an easy way and to visualize the results produced by the tools. An external database is used to store the design objects permanently.

RFT has been implemented in Lucid Common Lisp. Lucid Common Lisp allows Lisp programs to issue operating system commands and call functions written in other programming languages. These features have been used to integrate external analysis tools to the system. Because Lucid Common Lisp is supported by all major manufacturers of engineering workstations the above non-standardized features as well as graphics and windows functions are highly portable. RFT is currently running on Apollo, HP and Sun workstations.

OBJECT-ORIENTED TECHNIQUES AND TOOL INTERFACES

An important aspect of the RFT system is how the design tools are interfaced with the system to allow flexible communication between the tools and other parts of the system. Another aspect of this feature is the ease with which new tools can be integrated into the system.

The object-oriented implementation architecture of the system modularizes the system to a great extent. The architecture hides the internal details of the objects from other objects and all communication takes place with a well defined interface protocol. This makes the system highly modular and allows the development of tailored tools and configurations for different user groups. Also, modifications in the internal operation of some design tool require no modification in the other parts of the system. For instance, the block editor which is used to define the design diagrams is completely separated from the other tools. Using the basic block editor, or some specialized form of it created by inheritance, the user can access any combination of design tools.

The interfaces to the design tools have to be developed individually for each design tool. The tool interfaces consist of two parts. The declarative part, defined by expert users, specifies the rules and models used in the tool interface. The procedural part, developed by software experts, implements the functional operations needed in the interface. The drawback of this scheme is that the user-defined declarative knowledge of the interface is not enough but a tailor-made procedural part is also needed. The amount of work needed to integrate a complicated design tool, e.g. a simulator, into the system has been estimated to be about one man-month.

CONVERSIONS TO SIMULATORS

In order to analyze a design the following steps are needed:

* select a proper analysis tool

* build a model for the selected tool

* represent the model with a proper syntax

* interpret or postprocess the results

A simulator interface is usually assumed to take care of the last two tasks. Modelling languages and graphical representations of simulation results are the only services offered by most of the interfaces. Occasionally a model editor is available that allows the user to input a graphical representation of a model which is then converted into a suitable input format for the simulator.

In addition to features mentioned above the user needs simulator user support. Selection of parameter values and verification of the model correctness are important details that have to be taken care of. These kind of services are very important to the user since often he does not have enough experience to perform them by himself.

RFT provides these features by using knowledge describing the proper formation of simulator input. The knowledge includes the syntax description of the simulator input language, for example the number and order of parameters for each simulator block. In addition it contains heuristic rules that are used to suggest values for simulation parameters. When the user wants to start a simulation this knowledge is used first to verify that the model defined by the user does not contain any errors and then to generate the input file for the simulator.

MODEL GENERATION

The techniques described in the above section assist the user with the details of using a simulator. If the first two tasks, tool selection and model building, are also automated the user does not have to pay any attention to the execution of the analysis. He can work with familiar design concepts and ask the system to measure the value of some interesting variable. The analysis tools and models that compute the requested value are hidden from the user. This also allows new (and hopefully better) analysis tools to be integrated into the environment in a way that is transparent to the user.

The tasks of tool selection and model building are highly knowledge-intensive. Again we use predefined knowledge to carry out the tasks. In RF-design there is a set of specification measurements which are necessary to validate that the product fulfills the specification defined by national and international telecommunication agreements. Therefore, in this field, the basic analysis tasks are well defined and known in advance which makes the knowledge-based approach possible. The knowledge used is divided into measurement specific and element specific parts. Measurement specific knowledge defines which elements are relevant to the measurement, the tools to be used and a formula for combining their results. Element specific knowledge defines the models that are used to simulate the behavior of the elements.

The above tasks require different forms of knowledge. Therefore a uniform knowledge representation technique like the use of rules is not sufficient in our application. We have used Lisp macros to develop a definition language for conversion knowledge. This concept has been flexible enough to allow the design engineers themselves to define all the necessary knowledge.

AN EXAMPLE OF A SIMULATED MEASUREMENT

To illustrate the behavior of this system and the usage of knowledge in different conversions we present a simplified example of system level analysis based upon the use of TOPSIM-simulator.

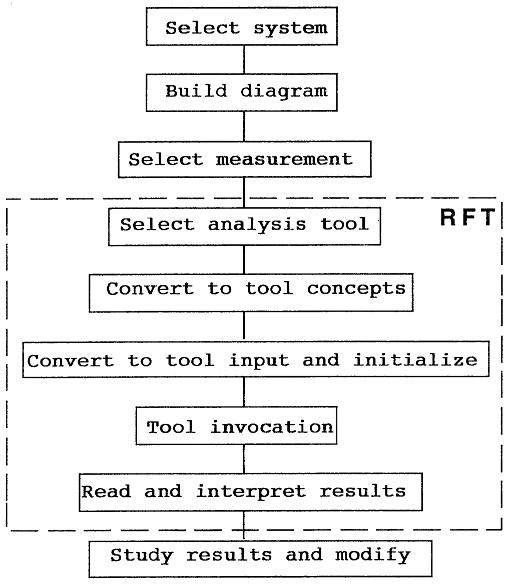

Figure 2. The subtask needed to perform a simulated measurement.

Fig. 2 represents the subtasks that are needed to perform a simulated measurement. The user inputs the design diagram and selects the measurement he wants the system to simulate. The routine tasks are carried out automatically by the computer. The computer also performs error checking during the whole operation in order to find possible design errors and thus avoid erroneous and misleading results.

The first task is to specify what kind of system is under development. The specifications of mobile telephones vary in different countries and in different networks. The selected system type defines requirements for the simulated measurements.

The actual design is done by the user, that is, he defines the topology of the design and specifies parameters for each block. In this phase the system already performs some crude error checking in order to notify the designer of possible errors as early as possible. However, comprehensive error checking is impossible since the user may decide not to define values for all parameters at this stage. As certain simulated measurements do not use all parameter values this is acceptable and errors from missing parameter values can only be found in later stages of the process.

```
(defsimulation Co_channel_rejection
   :target topsim
   :source source_1
   :measurements ((perspe :nfft    1000
                          :fscale  10.0
                          :kum     2)
                  (powmet :t0      0.1))
   :patterns-and-processing-rules
                ((;; pattern
                   (input mixer))
                 (;; pattern
                   (input output)
                  ;; processing rules
                   (band_pass_filter
                    limiter_amplifier
                    amplifier
                    fm_modulator))))
```

Figure 3. An example of measurement specific knowledge

Once the user has selected what kind of measurement he wants the system to perform it decides what is the necessary tool to do it. The definition of Fig. 3 specifies that the analysis is to be performed with the TOPSIM-simulator and defines what measurements the simulator should do. It also defines what kind of input signal is to be used and what components are relevant for the analysis.

The next step is to build the simulation model by converting the system level concepts to the corresponding simulator concepts. The conversion is not a one-to-one mapping. Often a number of ideal simulator blocks are needed to model the behavior of a non-ideal component. On the other hand, some functional level blocks can be irrelevant for the selected measurement and can be eliminated to simplify the model. Fig. 4 represents the conversion rule for a mixer and different models for its simulation.

The generated simulator model has to be further converted to a suitable input file for the simulator. During the conversion a number of additional checks are performed that concentrate on the dependencies between the parameter values and the signal types. For instance, TOPSIM requires some signals to have a particular type. The type of a signal is determined at run-time by a threshold value which depends on the simulation parameters. An illegal signal type leads to an error which is noticed only at run-time. To find this error before simulation, the values of signal types are estimated and propagated through the diagram to make sure that proper signal types are used.

```
;;; Conversion rule for Mixer
(defconversion Mixer
   (if block_type
     ;; simulate mixer using a table
     (progn (elements (bpnonl :a block_type
                              :bkoff 0.0)
                      (lomult))
            (connections (bpnonl lomult)))
     (if noise_figure
       ;; simulate noisy behavior
       (progn (elements
                 (lomult)
                 (nbwn :ix 1
                       :snr noise_figure
                       :f0 upper_rf_freq)
                 (bpsum :f0 upper_rf_freq))
              (connections (nbwn  bpsum)
                           (bpsum lomult)))
       ;; ideal mixer
       (elements (lomult)))))
```

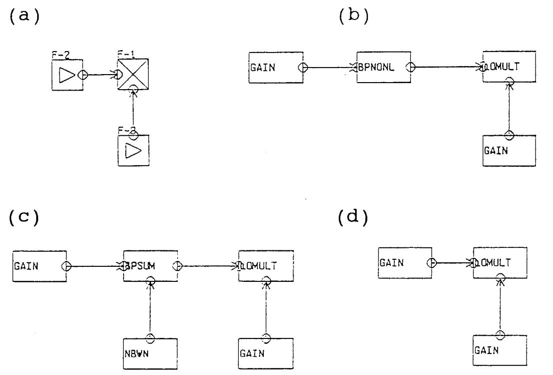

Figure 4. The use of a (simplified) conversion rule to convert mixer to TOPSIM-level. (a) original functional level diagram (b) TOPSIM-diagram of a mixer whose behavior has been measured (c) TOPSIM-diagram of a noisy mixer (d) TOPSIM-diagram of an ideal mixer.

Most of the knowledge needed for the conversion is stored in the definition of simulator blocks. An example definition of a simulator block ampmod is in Fig. 5.

```
(def-top-element ampmod
   :document "AMPMOD simulates an analog
             amplitude modulator."
   :class modem
   :inputs baseband
   :outputs analytic
   :attributes
    ((f0 nil frequency "carrier frequency")
     (amp 1 amplitude "carrier amplitude")
     (pha 0 phase "carrier phase in radians")
     (sens 1 am_sensitivity "sensitivity")
     (amean 1 voltage "DC component"))
   :constraints
       ("f0 > 1.0/(5.0*delt)"))
```

Figure 5. An example component knowledge definition

The block has 5 attributes: f0, amp, pha, sens and amean. Each attribute has a name, default value, attribute type and help text. The attribute types are defined in a separate file and consist of value ranges, default units and alternative unit fields.

Input and output type information is used to check the consistency of connected ports. The constraints field is used for error checking to find illegal parameter value combinations. Some components have additional keywords describing exceptions to the normal use of the block, such as signal type modification rules or reversed parameter order in the simulator input.

Once the conversion is successfully completed the simulator is started. When the simulation results are ready they are read back to RFT and displayed to the user in a suitable format.

As the above example shows the conversion is divided into two separate stages: conversion to the simulator level and conversion from the simulator level to the input syntax of the simulator. The simulator level representation can be manipulated through the graphical interface in the same way as the original functional level diagram. This organization has made it possible to use the simulator as a separate tool through the graphical interface. In specification measurements this feature is useful in the development of the conversion knowledge. Also this makes it possible for the users to operate with the simulator only. In that case they naturally need more expertise but can perform more flexible simulations.

DESIGN OPTIMIZATION AND RULE-BASED DESIGN SYNTHESIS

The above tools provide a way to analyze the design and verify whether or not it fulfills a given specification. They eliminate design routines and in that way encourage users to study more design alternatives. However, they do not give any hints of how to change the design to satisfy the specifications. A number of experimental systems have recently addressed this question, above all in VLSI design [8,9,10]. Unlike most of them we are not attempting to fully automate the design synthesis since complete automation of complex design tasks is not possible with current technology [11]. Instead, we provide the users some tools which suggest modifications to certain parts of the design. These synthesis tools are integrated into the design environment in the same way as the analysis tools. The goal of this organization is a cooperative system which couples the designer's judgement with computer's capabilities.

Design optimization

In RFT, mathematical optimization techniques are used to suggest values for some design parameters. The use of optimization algorithm as part of a knowledge-based system requires some properties from the algorithm:

i) The algorithm should be robust since the problem may not be properly formulated. Especially infeasible problems are often encountered and the system should be able to recover from these.

ii) The algorithm should be able to use other tools in the environment to compute the values of objective and constraint functions.

iii) The results of the algorithm do not have to be very accurate since they are viewed as guidelines. Moreover, highly accurate results are superfluous because of tolerances in manufacturing.

In RFT, we have used the simulated annealing algorithm [12] to solve the optimization problems. The benefits of this algorithm are that it is very robust and not easily mislead by local minima (requirement i). The implementation of the algorithm is simple and therefore is has been easily implemented as part of the RFT framework in such a way that it can use all services provided by the environment (requirement ii). This allows the values of the constraint equations to be computed using the normal analysis routines of RFT, and all modifications in analysis routines are automatically reflected to the optimization part as well. The main drawback of the algorithm is the large number of iterations needed which results in long execution times. However, the execution times can be reduced when highly accurate results are not needed.

Rule-based design

Another approach is to use expert system techniques. This is more general than optimization since no exact mathematical problem formulation is necessary. Here again the problem is the integration of the design rule-base with other parts of the system. In order to achieve the necessary flexibility we have not used any inference engine but compiled the rules to Lisp-functions. This approach of rule compilation allows us to complement the rules with arbitrary Lisp code. This is needed for procedural control structures and for the cooperation of the rule-base with the user interface and with other design tools.

So far only one small rule-base has been implemented with this feature. In this application rule-based design approach seems to be less useful than optimization since a large number of rules are needed for a relatively simple design task. To be really useful a large number of rule-bases of this size are needed, the development of which requires considerable work from the experts.

PRACTICAL EXPERIENCES

The system is currently in field-testing in several design groups at Nokia Corporation. Objective evaluation of the system performance is difficult since the quality of designs depends on many factors which are difficult to measure and compare. The same applies to design times since no studies were conducted before the system was delivered. The subjective experiences of the design engineers suggest major speed-ups in design times and an increase in the quality of the designs. The improvements result largely from elimination of routine tasks, from the detection of design errors in early stages of the design, and from the increased use of analysis and simulation tools.

When the system development and maintenance is concerned, several problems are evident. The integration of new tools into the design environment is not easy. The amount of work (about one man-month) and the need of a software specialist do not allow flexible enough integration of new tools and thus a continuous evolution of the environment. However, this seems to be a difficult problem especially when we do not want to set any strong requirements to the tools that can be integrated into the system. A partial solution to this problem would be a definition language that allows textual references to the topology of the design diagram. This feature would also be useful in the design rule-bases so that the rule-bases can access the design diagrams.

CONCLUSIONS AND FUTURE WORK

The above techniques have been applied in the development of a system for the design of mobile telephones. The resulting RFT-system offers easy access to different design tools as well as design synthesis options. The graphical interface of the system allows users to work with familiar design concepts and the tool-specific knowledge of the system takes care of necessary conversions. Although our application, RF-design, is a highly specialized field we believe that similar solutions could also be successfully used in other design and problem solving tasks.

The future work related to this research can be divided into two categories. First the continuous development of the basic framework and research on techniques that make the environment more easy to use and maintain. For instance, the use of object databases and techniques to facilitate the maintenance of the environment. Second, the development of new tools to be added into the framework. These include especially tools for design synthesis and for design diagnosis.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Nokia-Mobira for design expertise and funding. This work was also partially supported by Finnish Academy.

REFERENCES

1. Ketonen, T., Lounamaa, P. and Nurminen, J. K., "An Electronic Design CAD System Combining Knowledge-based Techniques with Optimization and Simulation," in Gero, J. S. (eds.), Artificial Intelligence in Engineering: Design, Computational Mechanics Publications, Southampton, U.K., 1988, pp. 101-118.
2. Nurminen, J. K., "Coupling Symbolic and Numeric Computing in Knowledge-Based Systems: An Application to Electronics Design," Licentiate's thesis, Systems Analysis Laboratory, Faculty of Information Technology, Helsinki University of Technology, 1989.
3. Ketonen, T. and Nurminen J. K., "An Intelligent Design and Simulation Environment," in Proceedings of the 1989 European Simulation Multiconference, Rome, Italy, June, 1989, pp. 201-205.
4. Ketonen, T. and Nurminen J. K., "Combining Symbolic Computing with Conventional Design and Analysis Techniques," in Proceedings of the Second Scandinavian Conference on Artificial Intelligence, Tampere, Finland, June, 1989, pp. 824-835.
5. Kitzmiller, C. T. and Kowalik, J. S., "Workshop report: Coupling Symbolic and Numeric Computing in Knowledge-Based Systems," AI Magazine, Summer, 1987, pp. 85-90.
6. "TOPSIM III User's manual", Torino Polytechnico, 1983,
7. Heikkilä, P., Valtonen, M. and Pohjonen H., "Automated Dimensioning of MOS Transistors without Using Topology-Specific Explicit Formulas," in the Proceedings of the 1989 ISCAS, Portland, Oregon, May, 1989.
8. Brewer, F. and Gajski, D., "An Expert-System Paradigm for Design," in proceedings of the 23rd Annual Design Automation Conference, June, 1986, pp. 62-68.
9. Mitchell, T. M., Steinberg, L. I. and Shulman, J., "A Knowledge-Based Approach to Design," IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-7(5), 1985, pp. 502-510.
10. Subrahmanyam, P. A., "Synapse: An Expert System for VLSI Design," IEEE Computer, July, 1986, pp. 78-89.
11. Mitchell, T. and Mostow, J., "Artificial Intelligence and Design," Sixth National Conference on Artificial Intelligence, Conference Tutorial Program, July, 1987.
12. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., Numerical Recipes The Art of Scientific Computing, Cambridge University Press, 1986.