TN 953763

Rec'd 6/17/11

79021136

STATUS: PENDING 20110616 OCLC #: 320287808

REQUEST DATE: 20110616 NEED BEFORE: 20110716 SOURCE: ILLiad

BORROWER: AZU RECEIVE DATE: DUE DATE:

RENEWAL REQ: NEW DUE DATE: SPCL MES:

LENDERS: *UF#, GEBAY, DKB, ODU, UF#

TITLE: Computer Aided Systems Theory - EUROCAST<<'>> 89 : Proceedings. A selection of papers from
the Internat. Workshop.. /
ISBN: 9780387522159
IMPRINT: Berlin : Springer, 1990.
SERIES: Lecture Notes in Computer Science ; 410.
ARTICLE: : Computer Aided Systems Theory and Knowledge-Based System Design and Simulation; Directions to Explore
ISSUE DATE: 1990
PAGES: 322-335
VERIFIED: <TN:953763><ODYSSEY:150.135.238.6/ILL> OCLC
SHIP TO: ILL/UNIVERSITY ARIZONA LIBRARIES/1510 E UNIVERSITY/TUCSON AZ 85721-0055

BILL TO: Same//FEIN 74-2652689//BLLD acct#51-105
SHIP VIA: Ariel, Odyssey or Library Mail
MAXCOST: IFM - $50
COPYRIGHT COMPLIANCE: CCL
ODYSSEY: 150.135.238.6/ILL
FAX: (520) 621-4619 //ODYSSEY PREFERRED/IL...
EMAIL: askddt@u.library.arizona.edu
AFFILIATION: AZNET ; GIF ; GWLA ; SHRS
BORROWING NOTES: For Book Chapter requests, scan the chapter only, do not lend book. If over your
page limit, please CONDITIONAL request. Thank you.

PATRON: Hwang, George

Lecture notes in computer Science
l - ES +

OCLC-Nr 00

Computer Aided Systems Theory and Knowledge-Based System Design and Simulation; Directions to Explore

Jerzy W. Rozenblit
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, Arizona 85721
U.S.A.

and

Herbert Praehofer
Dept. of Systems Theory and Information Engineering
Johannes Kepler University
A-4040 Linz, Austria

Abstract

This paper examines a possible merger of methods and techniques of Computer Aided Systems Theory (CAST) and Knowledge-Based System Design and Simulation Methodology. The basic tenets of both methodologies and the state of their implementation in computer-aided environments are discussed. The central focus of the paper is the application of CAST techniques to support the design model construction and development process.

1. Introduction

Progress in hardware and software technologies has resulted in availability of tools for the implementation of systems theory based concepts and frameworks (Pichler and Praehofer 1988, Praehofer 1986). Such frameworks are a basis for problem solving in a number of disciplines including system design and simulation modelling (Rozenblit and Zeigler 1988). In this paper we focus on the knowledge-based system design and simulation methodology, a framework derived from multifacetted modelling (Zeigler 1984). A brief discussion of CAST is followed by the description of our approach to system design and simulation. Then, we explore the synergism between CAST methods and our design methodology.

In examining the synergism, we pose the following questions: "What aspects of the design process can be supported by CAST?", "What system specifications can be made available by CAST and employed in system design?", "Can CAST support optimal design model development and selection?". We also briefly consider another set of problems. Such problems concern the question of whether we can incorporate the techniques (and their implementations) of knowledge-based design in CAST method banks.

The motivation for examining the above issues stems from the need to improve the design model specification process and extend our framework to include various classes of

modelling formalisms. CAST method banks can provide this type of assistance. We shall discuss this in detail in Sections 4 and 5. Since our design and simulation framework is based on the system theoretical approach, it is natural that we seek assistance offered by the computer-aided systems theory.

The contribution of knowledge-based design and simulation to CAST is viewed in the following context: the definition of CAST specifies it as a bank of interactive methods for problem solving (Pichler 1988, Pichler and Schwaertzel 1988). It is therefore worthwhile to consider creating method banks that would incorporate our modelling experience and tools in the CAST environment. Our experience in implementing such methods in Artificial Intelligence languages (Rozenblit et. al. 1988, Rozenblit and Huang 1987) may also prove useful for CAST realization.

In the ensuing section, we define the concepts of CAST that we plan to apply to our system design methodology. For a more detailed exposition of CAST research the reader is referred to (Pichler and Schwaerztel 1988, Pichler 1988).

2. Computer Aided Systems Theory

Systems Theory intends to provide general problem solving concepts for different fields of applications. Although Systems Theory knowledge is regarded as an important theoretical background for many technical and scientific disciplines and is included in many university curricula, the use of system theoretical methods for practical engineering is still low. The reason for that has been the lack of powerful computer implementations which would make Systems Theory easy to use (Pichler and Schwaertzel 1988).

Nowadays, with the availability of powerful workstations with user-friendly man-machine interfaces and the availability of modern software engineering concepts, Systems Theory should be given a new opportunity. By implementing system theoretical problem solving techniques in a user-friendly, easy to handle and easy to learn way, these techniques should become appealing to an engineer. Pichler (1988a) has defined an effort bringing Systems Theory to a domain termed Computer Aided Systems Theory.

This project, carried out at the Department of Systems Theory and Information Engineering at the University of Linz, aims to implement interactive method banks to support system theoretical problem solving. The underlying theoretical framework used to develop such method banks is STIPS (Systems Theory Instrumented Problem Solving) (Pichler 1986). The STIPS framework provides several system types, system transformations (analysis and synthesis operations) for manipulation of systems and the so-called STIPS machine (STIPS.M) to control the problem solving process. The STIPS problem solving process can be described in the following way: starting with an initial system description and using a control strategy imposed by the STIPS machine, new system specifications are derived. This process continues until a satisfying goal state is achieved. An implementation of STIPS—an interactive CAST method bank—has to provide schemes for computer representation of systems and system types, and modelling concepts for the system definition and implementation of system transformations.

3. Knowledge–Based System Design and Simulation

Our research employs Artificial Intelligence and Multifacetted Simulation Modelling to unify engineering design activities and develop a methodology for systematic simulation model construction and evaluation. The methodology is based on codifying appropriate decompositions, taxonomic, and coupling relationships. This constitutes the declarative design knowledge base. Beyond this, we provide the procedural knowledge base in the form of production rules used to process the elements in a design domain.

As a step toward a complete design knowledge representation scheme, we have combined the decomposition, taxonomic, and coupling relationships in a representation form called the *system entity structure*, a declarative scheme related to frame–theoretic and object–based representations. The entities of the entity structure refer to conceptual components of reality for which models may reside in the design model base. Also associated with entities are slots for attribute knowledge representation. An entity may have several aspects, each denoting a decomposition, and therefore having several entities. An entity may also have several specializations, each representing a classification of the possible variants of the entity. The generative capability of the entity structure enables convenient generation and representation of design model attributes at multiple levels of aggregation and abstraction. A complete specification of the system entity structure and its associated structure transformations are presented in (Zeigler 1984, Rozenblit and Zeigler 1988). We provide an illustrative example in Section 5.

The primary application of the above knowledge representation scheme is the objectives-driven development of design models. In this approach, a model is synthesized from components identified through the system entity structure and stored in the design model base. The synthesis process is guided by project's objectives, requirements, and constraints. The objectives guide the *pruning* process. The pruning process consists in specifying a knowledge base that contains rules for selection and configuration of the entities represented by the system entity structure. The designer invokes the inference engine which, through a series of queries based on the constraint rules, allows him/her to consult on an appropriate structure for the modelling problem at hand. The result is a recommendation for a design model composition tree (Zeigler 1984).

The model composition tree is a tree whose leaf nodes are system specifications. These are the atomic components which will be coupled in a hierarchical manner. The interior nodes n have the following specification attached to them: a system specification S_n, a coupling scheme C_n, and a morphism H_n. The coupling scheme C_n is used to interface the system specifications assigned to the children of the interior node. H_n establishes a correspondence between S_n and the resultant of the coupling process using C_n. The leaf nodes are assigned only system specifications which are atomic and are not subject to decomposition. The composition tree is used by DEVS-Scheme software environment (Rozenblit et. al. 1988, Zeigler 1987) to retrieve models from the model base. The retrieved models are automatically linked in a hierarchical manner according to the coupling constraints.

The modeling formalism used for system specification in our methodology is Discrete Even System Specification (DEVS) (Zeigler 1976, 1984). DEVS provides a formal

representation of discrete event systems. Formally, it is defined as follows:

DEVS is a structure:

$$M = < X, S, Y, \delta, \lambda, ta >$$

where:

X	is the external event set
S	is the sequential state set
Y	is the output set
δ	is the transition function
λ	is the output function
ta	is the time advance function

DEVS specifies a system

$$S = < T, X, \Omega, Y, \delta, \lambda >$$

where:

$T = Reals$
$X = X_{DEVS} \cup \{\phi\}$ (an empty event)
$\Omega =$ set of discrete event segment over X

The state set is defined as follows:

$$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$$

where:

$$ta: S \rightarrow R_{0,\infty}$$

and (s, e) is a total state pair, where s is a sequential state and e is elapsed time in state s.

The transition function consists from two pairs, namely:

$\delta_\phi: S \rightarrow S$ is the internal transition function

and

$\delta_{ex}: Q \times X \rightarrow S$ is the external transition function

The formal construction of the system's transition function δ is given in (Zeigler 1976). DEVS is closed under coupling. This property enables us to construct hierarchical DEVS network specifications. A detailed formal treatment of DEVS at the coupled system level is presented in (Zeigler 1984).

Performance of design models is evaluated through computer simulation in the DEVS-Scheme environment. DEVS-Scheme is an object–oriented simulation shell for modeling

and design that facilitates construction of families of models specified in the DEVS formalism. Models are evaluated in respective experimental frames. An *experimental frame* defines a set of input, control, output, and summary variables. Those objects specify conditions under which a model is simulated and observed. The environment supports construction of *distributed, hierarchical discrete event models* and is written in the PC-Scheme language which runs on IBM compatible microcomputers and AI Workstations.

We have been substantiating the above methodology by case studies involving design and simulation of distributed computer architectures, local area networks, and more recently, VLSI packages.

The modelling and simulation aspect of our methodology is currently focused on the discrete event domain. However, it is easy to notice that the entity structuring concepts are applicable to any class of systems that exhibit hierarchy and modularity of structure. Our first effort in improving the efficacy of the methodology is to extend the simulation concepts so that facilities for specification of systems other than discrete event will be available. We briefly describe these efforts in the next section.

4. Simulation Concepts Non–Homogenous Model Specifications

Although it is possible to emulate other types of dynamic systems, like differential equation specified systems or discrete time systems using the DEVS formalism, we plan to enrich the framework by developing simulation concepts for different system types. The following reasons motivate such a development:

1. One of the major achievements in developing the DEVS formalism and simulation concepts to simulate DEVS systems was to separate the formulation of the model from the implementation of the simulation program (Zeigler 1984). Employing DEVS, the user has to specify the model; the time scheduling during the simulation run is handled by the abstract simulator. This leads to a very clear and convenient model realization. DEVS models are mainly built for performance evaluation. When DEVS systems are used for simulation modelling where other types of systems would be more appropriate, a loss in modelling convenience may result. Parts of the time scheduling for the specific type of simulation, which could be done by a special abstract simulator, must be specified in the model. By using a specializes abstract simulator for each system type, the modelling process and its support may be improved.
2. An abstract simulator which uses all the implicit knowledge about the dynamic behavior of a special system type can be implemented in the most efficient way.
3. With a specific system type, we associate not only a specific dynamic behavior, but Systems Theory also provides system transformations whose applicability depends on the type of the system. Therefore the use of systems of the most appropriate type, most appropriate for the problem being solved, facilitates the use of system transformations. Thus, combining Computer Aided Systems Theory and Knowledge–Based System Design and Simulation becomes possible.

As a field of application for the different modelling and simulation concepts, we are considering VLSI hardware design. In this area, simulation is the major tool to evaluate and

verify design blueprints. Systems Theory methods and techniques are well developed and investigated in this domain as well. In hardware design, several different simulations and hence design models of different system types can be used. Differential equation specified systems are used to investigate physical properties at the transistor level. Differential equations are also used to specify and simulate the behavior of analogous devices. Discrete time models, networks of sequential machines, and networks of sequential circuits and Boolean functions can be used to design digital systems. They can also be used to evaluate the behavior of a design and to verify the functional design of a device without considering physical properties of its realization. DEVS systems can be used for logic simulation at the gate level. DEVS systems can be applied to evaluate the performance of a hardware system.

The simulation concepts will not only facilitate simulation of networks of components of the same type, but will also enable simulation of networks of components which are of different system types. We intend to develop simulation concepts for possible combinations of DEVS systems, differential equation specified systems and discrete time systems. These concepts will enable digital simulation of the following hardware devices:

1. hybrid systems (coupling of analog and digital systems, when the digital part is modeled by a discrete time system as well as when it is modeled at the gate level considering gate delay times)

2. asynchronous digital systems (couplings of digital systems which work with different clock times)

3. interrupts (can be modeled and simulated by coupling a DEVS model to a model of a digital device)

4. modelling and simulation of hardware devices where the parts of the device are modeled at different design levels (e.g., parts of the device are modeled at the gate level considering gate delay times, and other parts are modeled by discrete time systems).

The abstract simulator for a hierarchical DEVS has a hierarchical structure reflecting the structure of a hierarchical the model. For every atomic DEVS there exists a simulator, for every DEVS network there exists a coordinator (Zeigler 1984). We intend to built the abstract simulator for differential equation specified systems and discrete time specified systems in a hierarchical way. This simulator will reflect the structure of the model. As the abstract simulator for DEVS facilitates distributed simulation, so will the abstract simulators for the other types of system specifications.

A special coordinator has to be built for the simulation of networks of different type components. For each possible combination of system types, there must be a special coordinator. This coordinator has to coordinate the coupling between the parts of different type. The simulation of the parts can be done by the standard coordinators and simulators.

5. CAST Support for Knowledge–Based System Design and Simulation

We perceive CAST as being instrumental in supporting the development of simulation models. As we have stated in Section 2, the model development process begins with setting

up a system entity structure that generates model composition trees. The development of an entity structure is referred to as the static model structuring (Zeigler 1984a). The model static structure is defined by the entities specified in the system entity structure, the input, output, and state variables. Definition of transition and output functions adds the dynamic components to the model's specification.

We have developed software that enables us to set up and prune entity structures. However, there does not exist an explicit methodology which would guide the designer in successfully structuring a design problem using the entity structure concepts. Therefore, we would like to investigate principles and approaches for setting up design entity structures for a class of hierarchical, modular systems.

To set up the entity structure, the designer must conceptualize the domain in terms of the structure concepts. Thus, by examining these concepts, we can generate a set of questions that must be answered at every stage of the structuring process. For example, considering the concepts of aspect, multiple decomposition, and specialization, we have questions such as the following to be asked while defining an entity: what are the possible ways of decomposing this entity? Will it occur at most once or will there be a possibly varying number of like entities? Should this entity be treated as a generalized class with several sub-specializations?

Of course, not only must the designer be able to ask himself such questions, he/she must be able to answer them. To help on this side, we should develop criteria for making decisions of the sort elicited in the questions. For example, several decompositions for an entity are possible if, looking ahead one can foresee the existence of a set of atomic components that can be disjointly partitioned in more than one way. An entity should be considered as a generalized class, if its potential specializations possess a significant degree of common structure. Sometimes such criteria may refer to implications of decisions that may not be known until the structuring has progressed further. This kind of decision support can be provided by CAST and its transformation methods for various system specifications. We envision this support in the following way.

System transformations can be used to compute alternative dynamic structures for an entity, to compute realizations of an entity and decompositions of a given entity, provided a system specification is associated with it.

The major generic transformation that will be offered by CAST to support the entity structuring and model development are as follows (as a fundamental orientation and classification of system stansformations we use the approach of Pichler (Pichler and Schwaertzel 1988). As illustrations for the generic classes of transformations we use examples from the sequential machine theory):

A system transformation from the generic system type Black Box to the generic system type Generator is called a realization transformation. From an input/output relation of an entity, the dynamic behavior of the entity is derived. An example from the sequential machine theory is the machine identification problem.

Important are transformations which convert a system of type Generator into a another Generator. For a given dynamic structure, alternative dynamic structures are

obtained which can be better with regard to specific criteria. The sequential machine theory provides a number of such methods. Among them are: state minimization of sequential machines or linear machines, sequential circuit realizations of sequential machines, linear realizations or shift register realizations of sequential machines.

Another type of transformations are decompositions, transformations from the Generator type to the Network type. Decompositions not only affect the dynamic structure of an entity but also the static structure. A new aspect representing the decomposition will be added to the entity as decompositions are generated by CAST methods. The well known decomposition methods of the sequential machine theory are the classical decompositions employing the lattice of s.p. partitions of a machine. Gate realizations of sequential circuits or boolean functions are understood to be decomposition methods as well.

The reverse transformation, the computation of an atomic system from a network with identical dynamic behavior, is also important for model development. As DEVS, differential equation specified systems and discrete time systems are closed under coupling, this is always possible.

The conceptual idea for using CAST to support the model structuring process is depicted in Figure 1.

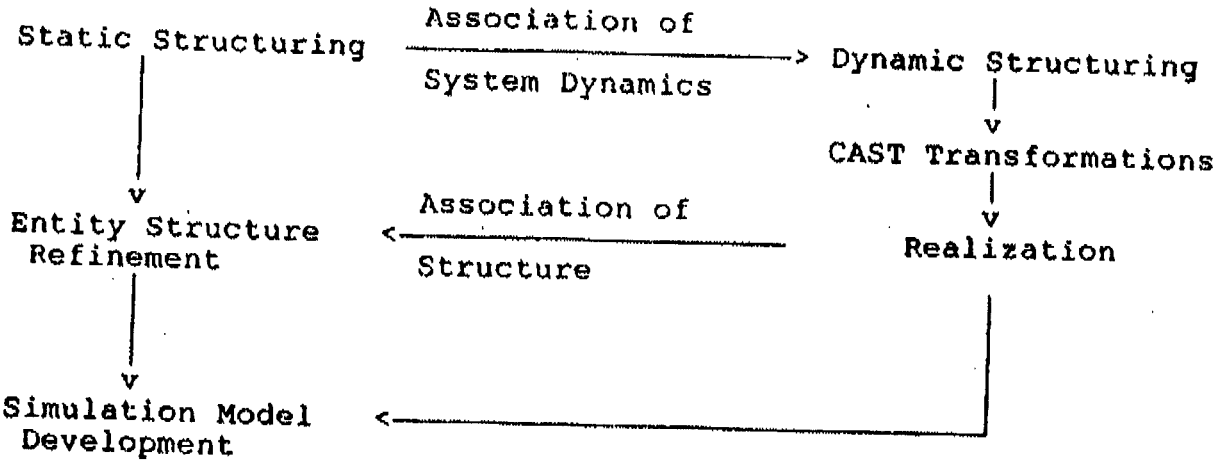

Figure 1. Model Structuring Support by CAST

At any point in the structuring process, the designer may face the types of questions we have mentioned above. He would then would invoke CAST transformations for generating

possible realizations of system specifications associated with a given entity. This result would serve as a basis for deciding whether or not and how to decompose the entity. For example, a CAST transformation may generate several implementations of a finite state automaton and the designer may use the concept of partitions with the substitution property to design circuits that require less components as a result of decomposition. There is another benefit of combining CAST methods with the entity structuring process. As CAST transformations are applied to systems specifications associated with the entities, simulation model specifications may be simultaneously generated and stored in the model base.

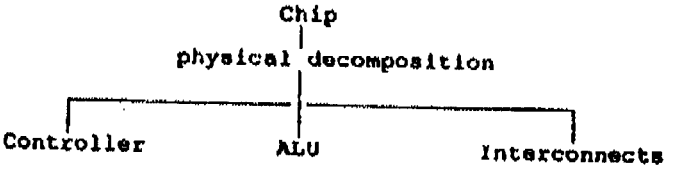

Figure 2a. System Entity Structure for Chip Design

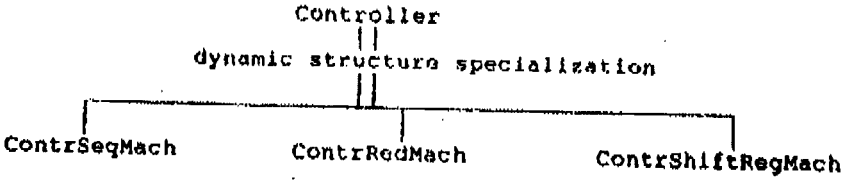

Figure 2b. Dynamic Structure Specialization Generated by CAST Classification of Dynamic System Specifications

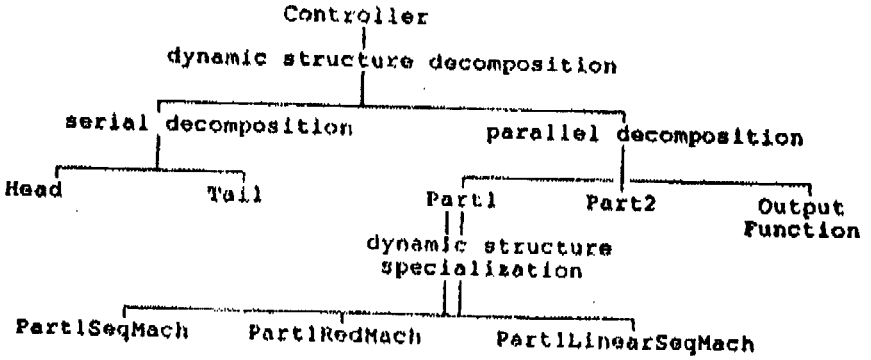

Figure 2c. Dynamic Structure Decomposition and Specialization Generated by CAST Transformations

A simple example demonstrates how the above concepts would be realized. Figure 2a shows a high level system entity structure for a chip design. The physical decomposition aspect represents a decomposition of the chip into its physical parts. As one of the parts, a controller is identified. The dynamic part of the controller can be represented by a sequential machine model, herein named "ContrSeqMach". By applying system transformations of the Generator to Generator type we obtain several alternative dynamic models of the controller, e.g., reduced machine (ContrRedMach) and shift register machine (ContrShiftRegMach) (Figure 2b). When we apply decomposition methods given by the CAST method bank to the controller's system specification, the result may be serial and paraller decompositions shown in Figure 2c. This figure also illustrates how the interactive application of CAST transformations can be extended into the next level in the system entity structure. The dynamic specification of "Part1" in parallel decomposition may be a a sequential machine "Part1SeqMach" or any one of other several alternative models derived through Generator to Generator transformations.

The major benefit of this approach is the ability to derive automatically model specifications for modelling domains for which CAST transformations are available, and the support in the entity structuring process. It is important to notice that in a complex design problem such model specifications may cut across several modelling formalism. Therefore, we have to undertake the efforts described in Section 4 of developing simulation concepts for non-homogeneous model specifications. For example, we may select a discrete time system to model the ALU unit and a sequential machine to model the controller. Therefore a special coordinator would have to be employed to simulate these two components at the coupled model level.

6. Current State of Implementations

We have undertaken efforts towards implementing the theoretical concepts underlying both the system design methodology and the CAST research.

6.1 Software for System Design and Simulation Support

To support the design process, we have available a set of software tools that are currently being integrated in a shell running on an AI workstation. The basic organization of the software is depicted in Figure 3.

The system entity structure specification has been incorporated in the DEVS-Scheme environment under the name ESP-Scheme. The program helps the modeller conceptualize and record the decompositions underlying a model, or family of models, before, during, and after development. To the extent that ESP-Scheme is used before beginning model development, it is a tool for assisting in top down model design. However, when additions and changes are made as the development proceeds, ESP serves as a recorder of progress. At the end of the development phase, the record constitutes de facto documentation of the system structure arrived at. Pruned entity structures serve as a basis for retrieval from a model base of model components specified in DEVS-Scheme. This is accomplished by the Transform procedures (Kim 1988, Zeigler 1987).

To aid in the pruning process, we have developed an expert system shell (Rozenblit

and Huang 1987, Rozenblit et. al 1988) which generates design model composition trees given a set of design constraints and requirements expressed as production rules.

The architecture of the DEVS-Scheme simulation system is derived from the abstract simulator concepts associated with the DEVS formalism. These concepts are naturally implementable by multiprocessor computer architectures. Therefore, models developed in DEVS-Scheme are readily transportable to distributed simulation systems design according to such principles.

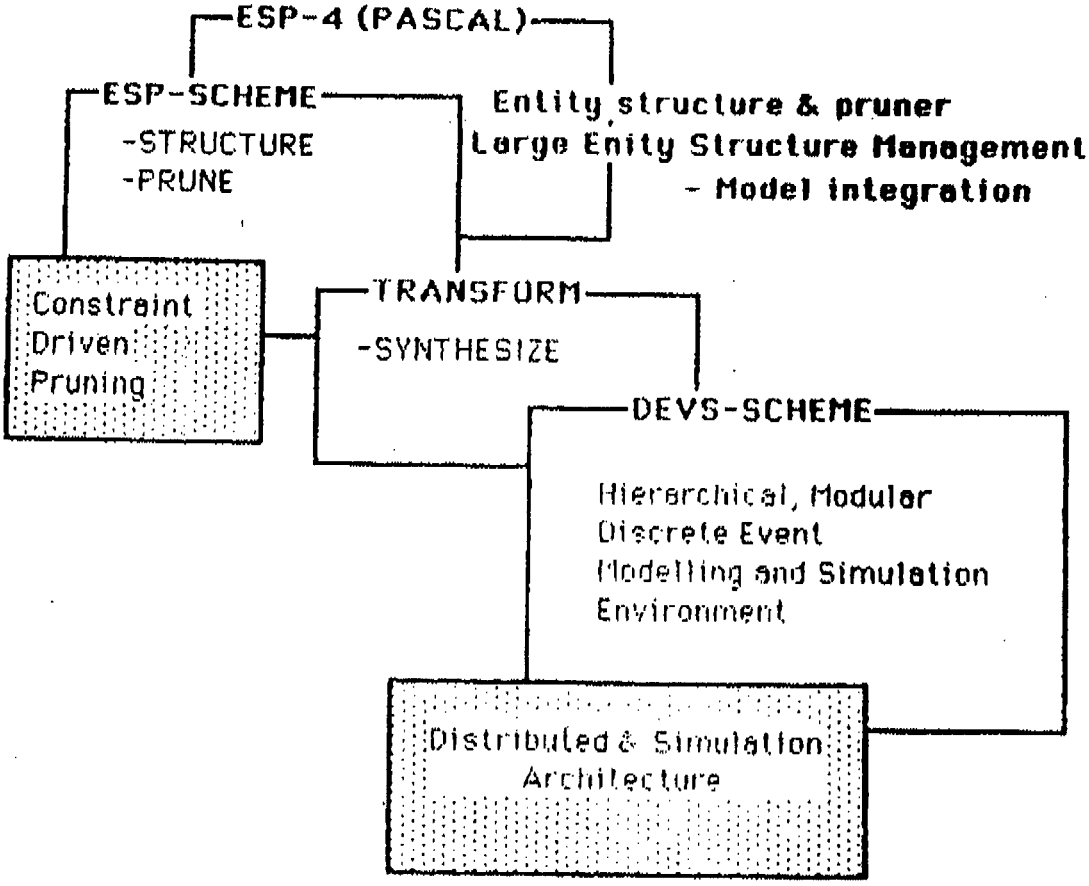

Figure 3. Organization of System Design and Simulation Support Software

DEVS-Scheme is coded in SCOOPS, an object oriented environment provided by PC-Scheme. All classes in DEVS-Scheme are subclasses of the universal class *entities* which provides tools for manipulating objects in these classes. The inheritance mechanism ensures that such general facilities need only be defined once. Entities of a desired class may be constructed by using a method *mk-ent* which makes an entity, and *destroy* which destroys an entity.

Models and *processors*, the main subclasses of entities, provide the basic constructs needed for modelling and simulation. *Models* are further specialized into *atomic-models* and *coupled-models* which in turn are specialized into more specific categories. This process may continue until the user builds up a desired model base. Detailed description of the class hierarchy in DEVS-Scheme is given by Kim (1988).

In this environment, the user, whether human or artificial, is a goal–directed agent who examines the knowledge base and synthesizes a simulation model.

6.2. Current State of CAST Implementation

The start of the project CAST at the University of Linz was the implementation of an interactive method bank to support Finite State Machine methods. The name of this system is CAST.FSM (Computer Aided Systems Theory: Finite State Machine) (Pichler 1988, Pichler and Praehofer 1988). It evolved out of the attempt to use sequential machine methods for design for testability problems in VLSI hardware design (Praehofer 1986). It is a prototypic implementation which is used to show the applicability of Finite State Machine methods to hardware design.

CAST.FSM is coded in the functional programming language Interlisp-D, a LISP dialect developed by XEROX Parc, and the object oriented superset of Interlisp-D LOOPS (Lisp Object Oriented Programming System). It runs on SIEMENS 5815 workstations which are equivalent to XEROX 1108 workstations. Interlisp-D is a powerful LISP dialect including a powerful programming environment. It emphasizes an interactive, display oriented programming style and therefore provides techniques to implement modern man/machine interfaces, like window, mouse and menu technique. These techniques have been used to implement a user-friendly, interactive interface for CAST.FSM.

For the structuring of the program, we used the object oriented programming paradigm of LOOPS. Using this paradigm, a natural computer representation of the Systems Theory knowledge was possible. System types are represented by class definitions. Systems are instances of classes and system transformations are implemented by method definitions.

The interactive method bank CAST.FSM is already well developed. Many different system types and a number of methods for system definition, system representation and system transformation are implemented. Among the most important system types are sequential machines, deterministic sequential machines, linear sequential machines, sequential circuits and different types of functions. Examples of implemented system transformations which also can be of interest in a knowledge–based system design and simulation environment are: reduction of deterministic and linear sequential machines, parallel and serial decomposition of deterministic machines using the lattice of s.p. partitions, the computation of a sequential circuit realization or the shift register realization of sequential machines.

Another part of CAST deals with Petri Nets. This program called CAST.PN (Mittelmann 1988) is implemented in the same environment as CAST.FSM. The design of the program is based on the object oriented programming paradigm and it uses the

interactive, graphic I/O facilities. A means to specify a Petri Net model interactively and graphically are provided. It is possible to associate arbitrary LISP functions with the event nodes, which are executed when the event is active. The user can choose between two types of models, condition/event systems and place/transition nets.

The program LISAS (Lisp Implemented Systolic Array Simulator) (Mueller 1986) can be regarded as another CAST implementation facilitating the simulation of synchronous cellular arrays. It is specially developed for the simulation and verification of systolic array designs. As the other CAST implementations, LISAS is coded in Interlisp-D/LOOPS. The user interface facilitates interactive design and simulation of the array. The nodes can be specified by LISP expressions which results in a high flexibility of the nodes' function definition.

The integration of software tools employed by CAST and the system design methodology will be simple due to the same programming paradigms used in their design (i.e., object oriented programming).

7. Conclusions

We have explored a potential application of CAST concepts to support knowledge-based system design and simulation methodology. We focused on the interactive application of system transformations to derive decompositions and realizations of a system in the model development process.

There are a number of other issues that we feel should be investigated in the context of merging the two developments. It is desirable to consider incorporating in the CAST method banks procedures for specification of design objectives, requirements, and constraints as well as schemes for model validation and simplification. In this venue, we plan to employ our simulation modelling experience with discrete event systems and extend the results to modelling formalisms discussed in Section 4. We plan to verify the concepts presented in this papers in case studies from the area of signal processing hardware design.

Acknowledgment

The ideas presented in this paper originated during discussions at the 1988 CAST Workshop, at Gallneukirchen, Austria, April 1988. The authors would like to thank Professor Franz Pichler for making it possible for them to participate in the workshop.

References

Kim, T. G., (1988) A Knowledge-Based Environment for Hierarchical Modelling and Simulation, Doctoral Dissertation, University of Arizona, Tucson.

Mittelmann, R. (1988) Object Oriented Implementation of Petri Nets Concepts in: *Cybernetics and Systems '88* (ed. R. Trappl) Kluver Academic Publisher, pp. 731 - 736

Mueller-W., T. (1986) LISP Implemented Systolic Array Simulator. Master Thesis (in German) University of Linz, Austria, 1986

Pichler, F. (1986) Model Components for Symbolic Processing by Knowledge-Based Systems: The STIPS Framework, in: *Modelling and Simulation Methodology in the Artificial Intelligence Era* (eds. Elzas, M. et. al.), North-Holland, Amsterdam pp. 133-142

Pichler, F. and H. Schwaertzel (1988) *CAST: Computerunterstuetzte Systemtheorie Konstruktion interaktiver Methodenbanken*, Springer Verlag Berlin (to appear)

Pichler, F. and H. Praehofer (1988) Computer Aided Systems Thoery: Finite State Machines in: *Cybernetics and Systems '88* (ed. R. Trappl) Kluver Academic Publisher, pp. 737 - 732

Pichler, F. (1988) CAST—Modelling Approaches for Software Design. *Proc of the Sixth Symposium on Empirical Foundations of Informations and Software Sciences (EFISS)*, October 19-21, Atlanta.

Pichler, F. (1988a) CAST–Computer Aided Sytems Theory: A Framework for Interactive Method Banks, in: *Cybernetics and Systems '88*, (ed. Trapl, R.) Kluwer Academic Publishers, pp. 731- 736

Praehofer, H. (1986) LOOPS Implentation of automata theoretical methods. Master Thesis (in German) University of Linz, Austria.

Rozenblit, J. W. and Zeigler, B. P., (1988) Design and Modelling Concepts, in: *International Encyclopedia of Robotics*, (ed. Dorf, R.) John Wiley and Sons, New York.

Rozenblit, J. W. and Zeigler, B. P., (1986) Entity Based Structures for Experimental Frame and Model Construction, in: *Modelling and Simulation in the Artificial Intelligence Era*, (ed. M. S. Elzas, et. al.) North Holland, Amsterdam, pp. 79-100.

Rozenblit, J. W. and Y. M. Huang (1987) Constraint-Driven Generation of Model Structures. *Proc. of the 1987 Winter Simulation Conference*, Atlanta, December, pp. 604-611

Rozenblit, J. W., Kim, T. G. and B. P. Zeigler (1988) Towards the Implementation of a Knowledge–Based System Design and Simulation Environment. *Proc. of the 1988 Winter Simulation Conference*, San Diego. December.

Zeigler, B. P. (1976) *Theory of Modelling and Simulation*, John Wiley and Sons, New York.

Zeigler, B. P. (1984) *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, London.

Zeigler, B. P. (1984a) System–Theoretic Representation of Simulation Models. *IIE Transactions*, March, pp. 19-34

Zeigler, B. P. (1987) Hierarchical, Modular Discrete Event Modelling in an Object Oriented Environment. *Simulation Journal*, vol 49:5, pp. 219-230.