Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

APPLYING KNOWLEDGE-BASED SYSTEM DESIGN AND SIMULATION IN INFORMATION SYSTEM REQUIREMENTS DETERMINATION

Kung-Chao Liu

Department of Management
Information Systems
The University of Arizona
Tucson, Arizona 85721

Jerzy W. Rozenblit

Department of Electrical and
Computer Engineering
The University of Arizona
Tucson, Arizona 85721

ABSTRACT

Knowledge-Based System Design and Simulation (KBSDS) is a general system design framework that uses system entity structure, rule-based expert system techniques, and hierarchical system modeling and simulation. KBSDS has been successfully applied in a number of areas. In order to appraise the applicability of KBSDS in information system development, a spectrum of paradigms for use in the early stages of information system development was carefully selected and compared with KBSDS. This paper reviews and compares KBSDS and the representative paradigms of information system requirements determination. The comparisons favor the usability of KBSDS in information system requirements determination.

1. INTRODUCTION

Knowledge-Based System Design and Simulation (KBSDS) is a general system design framework emerging from artificial intelligence and system simulation [Rozenblit and Zeigler 1988]. The instruments used in KBSDS include *system entity structure* (SES), rule-based expert system techniques, and hierarchical system modeling and simulation. The SES represents the elements of a system under study. The rule-based expert system technique is used for selecting design configurations of the system. Hierarchical system modeling and simulation is employed to model and evaluate dynamic behavior of the system.

The major activities involved in the KBSDS design process are [Rozenblit and Zeigler 1988]: (1) specification of the SES representing the structure of the problem, (2) formulation of the production rules for use in an expert system to prune the SES according to design constraints, (3) transformation of the pruned SES into a hierarchical system model suitable for system simulation, and (4) system simulation and evaluation of the results to choose the most appropriate design.

KBSDS has been applied in a number of studies [Rozenblit and Zeigler 1988; Sevinc and Zeigler 1988; Zeigler et al. 1988]. The results have shown that KBSDS is a sound conceptual basis for integrated, model-based system development in the computer engineering domain.

KBSDS has also been used in a research project in development of information systems [Herniter et al. 1989]. In particular, Liu [1990] used KBSDS as the basis of an environment for building and evaluating prototypes of information systems modeled in the data flow model. Preliminary results from Liu's work indicate that KBSDS could become a basis for theoretic concept development in the domain of information system development.

Given the initial success of using KBSDS in developing information systems, the question of the general applicability of KBSDS in the domain of information system development arises. The purpose of this paper is to investigate this question. We focus on the early stages of information system development to keep our work within reasonable complexity. The early-stage work has been called requirements determination. Our process consists of selecting a spectrum of paradigms for information system requirements determination and then comparing them with KBSDS to assess the utility of KBSDS in the information system development domain.

In the following sections, we first review KBSDS, then outline those representative information system development paradigms, and finally describe the comparisons of the two to show what information system developers can and cannot expect from KBSDS. We also discuss what developers can do to enhance KBSDS. Since KBSDS is a hybrid of artificial intelligence and system simulation, the comparisons in a way assert the applicability of both artificial intelligence and system simulation in the field of information system requirements determination.

2. KNOWLEDGE-BASED SYSTEM DESIGN AND SIMULATION (KBSDS)

The KBSDS approach consists of three major conceptual elements: the system entity structure, rule-based pruning, and the *DEVS* formalism for discrete-event system modeling and simulation.

2.1 System Entity Structure

A *system entity structure* (SES) represents the components of a system and their *decompositions*, *taxonomies*, and *couplings* [Zeigler 1984; Rozenblit and Zeigler 1988]. The components are called entities, which have associated variables for representing their attributes. 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 taxonomy of the possible variants of the entity. Therefore, decompositions and taxonomies can occur alternately to form large SESs. Besides, aspects can have coupling constraints attached to them. Coupling constraints restrict the way in which entities under the aspects can be joined together.

With the SES representation, designers can depict their perception of the system under study. They record how the system can be broken into components, how some components are substitutes for each other, and how components can be related.

2.2 Rule-Based Pruning

Besides the declarative knowledge represented in SES, designers specify a set of production rules for manipulating the entities in a SES. These rules encode explicitly the criteria for selecting entities and synthesizing appropriate system configurations from selected entities.

Designers do not execute the *selection rules* and *synthesis rules* themselves. Instead, they define a knowledge base employed by a rule-based expert system [Rozenblit and Huang 1987]. Output from the expert system helps designers to determine a tailored SES that represents a system configuration fulfilling all the selection and synthesis criteria. The process of obtaining such a specific SES has been called *pruning* [Rozenblit and Zeigler 1988].

2.3 DEVS

DEVS is a formalism for discrete-event system modeling and simulation [Zeigler 1984]. Each system described in DEVS is called a *model*. An *atomic model* is a self-contained state transition machine that consists of timing functions, internal and external state transition functions, and output functions. The elegance of DEVS is in that one can customize an atomic model by tailoring or totally disabling any of the functions while still maintaining the modularity of the model.

Because of the modularity of DEVS models, one can construct a *composite model* from atomic models or other composite models in a hierarchical manner. Therefore, a model of a system can be simply an atomic or a composite model type. Specifically, one can realize all the terminal-level entities in a pruned SES as atomic models and all the non-terminal-level entities as composite models. The composite model for the root entity is the model of the system being studied.

Experimental Frame (EF) [Zeigler 1984] is a specific type of a composite model that manages the simulation settings for a model under study. An EF is composed of three atomic models: *Generator*, *Transducer*, and *Acceptor*. A generator generates events for the model being simulated. A transducer gathers statistics from the model being simulated. An acceptor monitors the statistics and terminates a simulation run according to pre-defined conditions. In essence, an EF encodes performance measures used to evaluate the system under study.

In a simulation environment, the EF and the model to be simulated are coupled to form a higher-level composite model. The higher-level composite model is executed by a simulation run-time system [Zeigler 1987]. Rozenblit [1985] has shown that distributed use of EFs is more desirable than centralized use. Since each EF realizes some design objectives, designers benefit more from having an EF for each component of the model of the system under study.

Zeigler [1987] has implemented DEVS in Scheme. The resultant *DEVS-Scheme* environment is a good platform for system design. Designers can map DEVS models into equivalent DEVS-Scheme program segments in a straightforward manner. Having specified the programs for the terminal-level entities in a pruned SES, the designers can synthesize the program for the whole system by level.

2.4 Design Phases

Given the instruments of KBSDS, a system designer follows a five-stage procedure to develop system models:

1. **SES Definition**. Specifying the SES that represents the designer's perception of the problem domain and possible configurations of the system to be developed.
2. **System Instance Definition**. Formulating selection and synthesis criteria in production rules and using them in an expert system to prune the SES. The result is an SES instance representing a configuration of the system under study.
3. **Simulation Model Definition**. Composing the DEVS model in a hierarchical manner directed by the pruned SES. EFs are added into every atomic and composite model for managing the simulation process.
4. **System Simulation**. Running simulation and gathering the information as specified in EFs.
5. **Evaluation**. Analyzing the simulation results and deciding whether to go back to any of the previous stages and modify model or EF components.

Table 1 summarizes the KBSDS approach to system development.

3. INFORMATION SYSTEM REQUIREMENTS DETERMINATION

Liu and Purdin [1990] investigated information system developers' work and identified a development process that consists of four stages: elicitation, specification, elaboration, and animation. Elicitation is employed to acquire from the customer all the facts related to the required system. Specification formalizes and specifies the system requirements in a clear and consistent manner. Elaboration realizes the requirements. Animation demonstrates the realization of the system for the customer.

In the research reported here, we focus on the requirements determination work, that is, the elicitation and specification stages, of gathering all the facts about the system to be developed and collating them into a set of well organized specifications. The matters of concern during requirements determination are: what to specify, how to specify, and how to derive the requirements [Yadav et al. 1988].

3.1 What to Specify

The question of what to specify is about what constitutes the requirements specification for an information system. Davis [1988b] proposes a specification of three levels: *user needs*, *solution space*, and *external behavior*.

User needs are the general information about the problem at hand. The solution space is the set of all possible solutions. Customers, developers, and others may all have some constraints on the system to be developed. The constraints reduce the number of possible solutions. Therefore, the solution space is not specified

Table 1. Summary of the KBSDS Approach

Stage	Activities	Instruments	Results
SES Definition	Defining generic SES	SES	SES representing problem domain
System Instance Definition	Formulating selection and synthesis criteria; Pruning SES	Expert system	Selection and synthesis rules; SES representing system configuration
Simulation Model Definition	Developing EFs; Composing DEVS model	DEVS; EF	Composite DEVS model (with EFs) representing the system
System Simulation	Running simulation; Gathering information	DEVS-Scheme	DEVS-Scheme program; Observed data
Evaluation	Analyzing simulation results		Decision to iterate or not

directly, but in terms of the constraints given. External behavior is defined when the solution space is known and a specific solution is chosen. (The term "external behavior" actually means everything about a certain system as perceived from the user's viewpoint.)

3.2 How to Specify

The question of how to specify is about the way of representing the requirements specification. Davis [1988b] recommends using Yeh and Zave's [1980] principles of *partitioning*, *abstraction*, and *projection* in all three levels of descriptions. Projection is a means of reflecting a particular view of the system. Abstraction refers to the process of focusing on features of the system that are salient to a discussion, and then adding successive layers of detail to the specification. Hierarchical and functional partitioning are viewed as examples of abstraction.

In particular, the technique proposed by Harel et al. [1988] receives high scores in Davis' [1988a, 1988b] evaluation of techniques for describing external system behavior. This technique describes a system in three aspects: *structural*, *functional*, and *behavioral*. These aspects are the hierarchical decompositions of a system's physical components, functional activities, and control activities, respectively. A functional activity is carried out by some physical component and regulated by some control activity. This relationship binds the three complementary aspects to form a complete system description.

3.3 How to Derive

The question of how to derive the requirements focuses on the way developers generate the requirements specification. This is largely a matter of design problem solving [Carroll et al. 1979]. Adelson and Soloway [1985] have synthesized a framework that describes software developers' design problem solving behaviors. These behaviors are: forming a mental model, systematically expanding the mental model, simulating the mental model, making notes, representing constraints, and labeling and retrieving models.

When designers are given a problem, they first form mental models of how they perceive the problem. A mental model is usually something that helps to decompose the problem into smaller ones and helps to organize those small parts together. Designers expand their mental models into more and more details as time elapses. From time to time, designers simulate mental models to check whether they really represent solutions to the problem. Note making helps designers to expand mental models systematically.

When the designers are not familiar with a (sub-)problem, they represent constraints explicitly to clarify it and subsequently decompose it. When they encounter a familiar (sub-)problem, they mark it with the label of a retrievable model.

3.4 Summary

In developing information systems, the developers first form a model of the system under study and gradually expand the model to a certain degree of detail. During the process, if the developers encounter unfamiliar issues, they explicitly study the constraints. If they encounter familiar issues, they just apply old models they have accumulated. From time to time, the developers may simulate the model to validate it.

The contents of the model include statements of the problem, all the constraints that have surfaced, and the descriptions of a system that fulfill all the objectives and constraints. The descriptions of the system have structural, functional, and behavioral facets.

4. THE USE OF KBSDS IN INFORMATION SYSTEM REQUIREMENTS DETERMINATION

The mapping between KBSDS and the works of Davis, Harel et al., and Adelson and Soloway are in Tables 2, 3, and 4, respectively.

Observing the comparisons presented in Tables 2, 3, and 4, we see more similarities than differences. The major difference is that KBSDS always leads to the development of constraints, which are the combination of SES, attached variables, and selection and synthesis rules. In other word, the KBSDS approach requires that constraints be developed before the solutions could be derived. This way of explicitly handling constraints shapes KBSDS into a more solid approach to system development.

However, developers have to consciously handle the process of gradual expansion. Currently KBSDS does not provide computer-aided tools to fully automate the iteration process because human judgment is required in analyzing simulation results.

Based on the information system requirements determination process portrayed in the previous section and the KBSDS process summarized in Table 1, here is how information system developers may use KBSDS. They first build an SES denoting the problem under study and encode all the constraints in the form of attached variables and selection and synthesis rules. Pruning mechanisms help the developers choose an admissible configuration of the system, which is one solution to the problem. After this, the developers proceed to specify the functional and behavioral descriptions in DEVS models. Whenever applicable, they may retrieve models from a model base instead of specifying every detail of each

Table 2. Davis [1988b] versus KBSDS

Davis	KBSDS
Partitioning, abstraction, and projection	Hierarchical and functional partitioning are instances of abstraction, which can be expressed through decomposition relation of SES. Projection can be expressed through the aspect notion offered by SES. Decomposition and aspect are governed by the axioms of SES.
User needs	SES contains items of user needs. In some cases, extensions to SES [Hu et al. 1989] may be desired.
Solution space	SES, attached variables, and selection and synthesis rules form the constraints. Pruning mechanisms help to reduce the solution space.
External behavior	External behavior is the combination of pruned SES, composite DEVS model, and EFs; see Table 3.

Table 3. Harel et al. [1988] versus KBSDS

Harel et al.	KBSDS
Structural aspect	SES can represent hierarchical decompositions of physical components. Each component is a DEVS model. Information flows between components are couplings between DEVS models.
Functional aspect	SES can represent hierarchical decompositions of functional activities. Each function is a DEVS model. Data flows between functions are couplings between DEVS models; see Liu [1990] for details.
Behavioral aspect	SES can represent hierarchical decompositions of control activities. Each control is a DEVS model. Events between controls are couplings between DEVS models.
Relationship between functional and structural aspects	Axioms of SES handle this.
Relationship between functional and behavioral aspects	Axioms of SES handle this.

Table 4. Adelson and Soloway [1985] versus KBSDS

Adelson and Soloway	KBSDS
Forming mental model	SES, production rules, DEVS models together is the representation of the system. This helps designers to organize their ideas explicitly.
Expanding mental model	This is implied in the iteration of the KBSDS procedure.
Simulating mental model	KBSDS helps designers to run the simulation mechanically. This should provide more insights than running simulation mentally.
Making notes	This is implied in the iteration of the KBSDS procedure. Annotations to SES and DEVS models may be necessary.
Representing constraints	See "Solution space" in Table 2.
Labeling and retrieving models	A base of DEVS models and their retrieving mechanisms [Rozenblit et al. 1990] facilitate reuse of models. Axioms of SES also help.

model. The developers may verify the specifications by performing simulation in DEVS-Scheme.

5. CONCLUDING REMARKS

We have compiled, from a set of seminal literature, the answers to "what to specify, how to specify, and how to derive" information system requirements during the early development stages. We have compared KBSDS to this normative situation and shown that KBSDS supports most of the activities happening therein. In particular, the capabilities for specifying system requirements in a theoretically supported means, accumulating model bases, and simulating the system specification are found lacking in other approaches.

To use KBSDS more effectively, information system developers may want to tailor KBSDS. A good example is what Liu [1990] has done to establish an environment for prototyping information systems in the data flow model. He set up generic SES for data flow modeling and suggested several sub-SESs that could be attached to the generic SES to fit different purposes. Used with an appropriate DEVS-Scheme model base, this prototyping environment is a prototype of a computer-aided "turn-key" system for information system development.

Beyond this, we envision a future picture of the KBSDS approach to information system development: (1) There will be more KBSDS-based computer-aided system development environments, and (2) more specialized tools, such as graphical interfaces, will be added in for further customizing the KBSDS approach.

ACKNOWLEDGMENTS

The Department of Management Information Systems at The University of Arizona has provided K. C. Liu the resources for conducting this research. The work of J. W. Rozenblit has been partially supported by McDonnell Douglas Corporation.

REFERENCES

Adelson, B. and E. Soloway (1985), "The Role of Domain Experience in Software Design," *IEEE Transactions on Software Engineering SE-11*, 1351–1360.

Carroll, J.M., J.C. Thomas, and A. Malhotra (1979), "Clinical-experimental Analysis of Design Problem Solving," *Design Studies 1*, 84–92.

Davis, A.M. (1988a), "A Comparison of Techniques for the Specification of External Behavior of Systems," *Communications of the ACM 31*, 1098–1115.

Davis, A.M. (1988b), "A Taxonomy of the Early Stages of the Software Development Life Cycle," *The Journal of Systems and Software 8*, 297–311.

Harel, D., H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring (1988), "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," In *Proceedings of the 10th International Conference on Software Engineering*, 396–406.

Herniter, B.C., K.C. Liu, and M.O. Pendergast (1989), "Using Artificial Intelligence Based System Simulation in Management Information Systems Research: Three Case Studies," In *Advances in AI and Simulation: Proceedings of the Society for Computer Simulation Multiconference on AI and Simulation*, R. Uttamsingh and A.M. Wildberger, Eds. The Society for Computer Simulation International, San Diego, CA, 147–152.

Hu, J., Y. Huang, and J.W. Rozenblit (1989), "FRASES—A knowledge representation scheme for engineering design," In *Advances in AI and Simulation: Proceedings of the Society for Computer Simulation Multiconference on AI and Simulation*, R. Uttamsingh and A.M. Wildberger, Eds. The Society for Computer Simulation International, San Diego, CA, 141–146.

Liu, K.C. (1990), "An Environment for Information System Prototyping: A System Simulation Approach," In *Advances in Computing and Information: Proceedings of International Conference on Computing and Information, ICCI'90*, S.G. Akl, F. Fiala, and W.W. Koczkodaj, Eds. Canadian Scholars' Press, Toronto, Canada, 236–239.

Liu, K.C. and T.D.M. Purdin (1990), "A Generational View of Information System Development," Manuscript submitted for publication.

Rozenblit, J.W. (1985), "Experimental Frames for Distributed Simulation Architectures," In *Distributed Simulation 1985: Proceedings of the Conference on Distributed Simulation 1985*, P. Reynolds, Ed. The Society for Computer Simulation, La Jolla, CA, 14–20.

Rozenblit, J.W., J. Hu, T.G. Kim, and B.P. Zeigler (1990), "Knowledge Based System Design Environment: Foundational Concepts and Implementation," *Journal of Operations Research Society*, in press.

Rozenblit, J.W. and Y.M. Huang (1987), "Constraint-driven Generation of Model Structures," In *Proceedings of the 1987 Winter Simulation Conference*, A. These, H. Grant, and W.D. Kelton, Eds. 604–611.

Rozenblit, J.W. and B.P. Zeigler (1988), "Design and Modeling Concepts," In *International Encyclopedia of Robotics Applications and Automation*, Wiley, New York, 308–322.

Sevinc, S. and B.P. Zeigler (1988), "Entity Structure Based Design Methodology: A LAN Protocol Example," *IEEE Transactions on Software Engineering 14*, 375–383.

Yadav, S.B., R.R. Bravoco, A.T. Chatfield, and T.M. Rajkumar (1988), "Comparison of Analysis Techniques for Information Requirement Determination," *Communications of the ACM 31*, 1090–1097.

Yeh, R.T. and P. Zave (1980), "Specifying Software Requirements," *Proceedings of the IEEE 68*, 1077–1085.

Zeigler, B.P. (1984), *Multifacetted Modelling and Discrete Event Simulation*, Academic, New York.

Zeigler, B.P. (1987), "Hierarchical, Modular Discrete-event Modelling in an Object-oriented Environment," *Simulation 49*, 219–230.

Zeigler, B.P., F.E. Cellier, and J.W. Rozenblit (1988), "Design of a Simulation Environment for Laboratory Management by Robot Organizations," *Journal of Intelligent and Robotic Systems 1*, 299–309.