Applied Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713191765

DESIGN FOR AUTONOMY: An Overview

Jerzy W. Rozenblit[a]
[a] Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona

To cite this Article Rozenblit, Jerzy W.(1992) 'DESIGN FOR AUTONOMY: An Overview', Applied Artificial Intelligence, 6: 1, 1 — 18
To link to this Article: DOI: 10.1080/08839519208949939
URL: http://dx.doi.org/10.1080/08839519208949939

DESIGN FOR AUTONOMY: An Overview

JERZY W. ROZENBLIT
Department of Electrical and Computer Engineering,
University of Arizona, Tucson, Arizona 85721

This paper discusses desiderata for support of high-autonomy systems design. Knowledge-based design techniques are presented. Requirements for high autonomy are defined, and a design methodology for achieving them is described. The suggested techniques provide high-level aids for developing architectures and integrating high-autonomy systems with environments in which they are developed. The design methodology presented here stems from a multifaceted stimulation-modeling framework.

INTRODUCTION

Autonomy as a design goal can be defined as the ability of a system to function independently, subject to its own laws and control principles. Whereas achievement of full and complete autonomy in artificial systems still borders on the realm of impossible, quick strides are being undertaken to achieve high autonomy in engineering designs, as evidenced by recent research and development of high-autonomy systems (Antsaklis et al., 1989; Erickson and Cheeseman, 1986; Fishwick et al., 1991; Luh and Zeigler, 1991; NASA 1985; Rozenblit and Zeigler, 1991; Zeigler, this issue). Work in *high autonomy* stems, to a large extent, from NASA's Space Station program and its Systems Autonomy Demonstration Project (Erickson and Cheeseman, 1986; NASA, 1985). This project focuses on research in artificial intelligence (AI), human factors, and dynamic control systems in support of Space Station automation and robotics technology (Erickson and Cheeseman, 1986; NASA, 1985). The design, construction, and evaluation of an intelligent autonomous system shell was recognized as an important goal of the systems autonomy research.

Although most AI and expert systems (ES) tools and methods have been successfully applied to planning, scheduling, diagnosis, and control, the applications treat these functions as separate entities. A salient requirement in highly autonomous system is that such, and similar, functions can be integrated to support the operation of a complete system. Architectures that foster the integration have been proposed by a number of authors (Antsaklis et al., 1989; Luh and Zeigler, 1991; Sardis, 1983; Zeigler and Chi, 1990). The common basis for the proposed designs is *automatic intelligent control.* The "center of mass" of this work concentrates around the design of intelligent autonomous control systems; this trend is well justified as the control community has well-defined, theory-based methods that support such designs. At the same time,

Applied Artificial Intelligence: 6:1–18, 1992

the need to incorporate modeling and high-level decision-making techniques for reasoning under uncertainty in autonomous controllers is well recognized (Antsaklis, et al., 1989).

New approaches to design of high-autonomy systems have emerged that take a distinct, simulation-modeling approach. While most of the new developments are application oriented, they do provide building blocks that may eventually evolve into a comprehensive methodology for autonomous systems design. This special issue on Design for High Autonomy of Applied Artificial Intelligence brings together articles that present the emerging new research. The article by Tamburino and Rizki offers a technique termed *performance-driven autonomous design* for development of pattern-recognition systems. They propose a closed-loop system that consists of a pattern-recognition module and an automatic design subsystem. The space of design solutions is defined by the pattern-recognition model. This model also serves as the basis for evaluating the performance of a particular design. The design module combines various learning methods to generate a recognition system.

Smith and Goldberg use the production-rule formalism to develop a learning classifier system (LCS) as an approach to reinforcement learning problems. The LCS can be thought of as an automaton capable of varying its structure. Such systems can improve our understanding of automatic adaptation and thus could prove useful in design and control where high autonomy is required.

A constraint-based framework for achieving flexible autonomy in multiagent systems is proposed by Evans et al. In this approach, a model of distributed problem solving is developed in which coordination of agents—each having a problem-solving capacity—is defined as a constraint-satisfaction planning problem. A simulation-modeling study of an automotive repair facility serves as an experimental testbed.

An intelligent design-flow management technique developed by Bretschneider and Lagger improves computer-aided design (CAD) systems by eliminating the need for highly trained CAD operators, reducing the design time and cost as well as the number of design errors. The technique is based on modeling the design flow using Petri net formalism and employing rule-based problem solving. Although this work is intended mainly to make the CAD systems more autonomous, its tenets can serve as guidelines for a general design methodology.

Zeigler, in this issue as well as in Zeigler (1990) and in Zeigler & Chi (1990), formulates model-based concepts for high-autonomy systems. The key paradigm in this approach is the ability of the architecture to support multifaceted modeling of systems. Partial models are employed to represent distinct functions and objectives. Multifaceted-modeling methodology (Zeigler, 1984a) is employed to manage the multiplicity of models, their representations, and levels of abstractions and formalisms.

In this paper, we attempt to establish desiderata for high-autonomy system

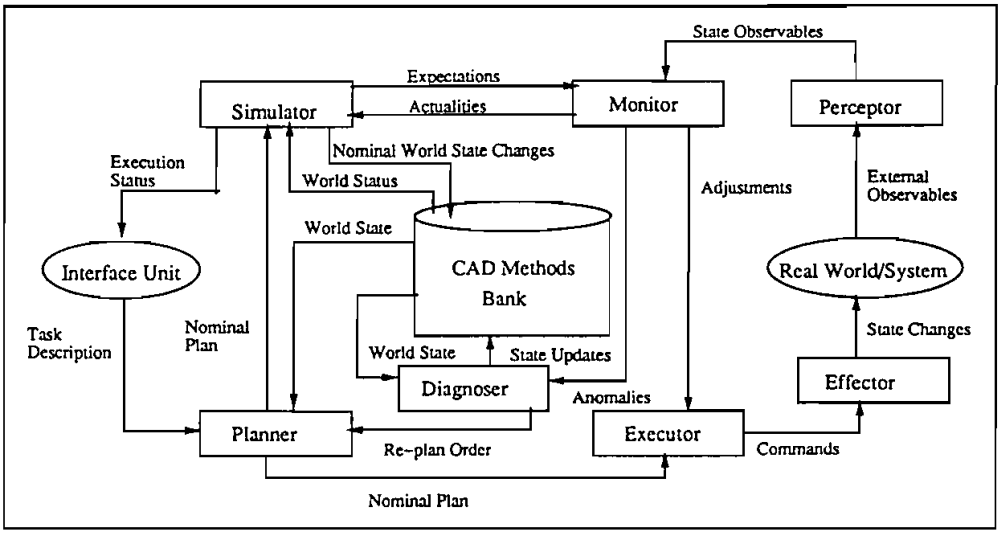

FIGURE 1. Intelligent autonomous architecture (adopted from Erickson and Cheeseman, 1986, and Zeigler, this issue).

design methodology. We emphasize the importance of integrating an autonomous architecture design with the design of the environment in which the architecture is to be deployed. In the ensuing section, we briefly summarize the requirements for high autonomy. Then, we discuss knowledge-based system design techniques. The techniques provide high-level aids for developing architectures of highly autonomous systems. The design methodology presented here stems from the multifaceted simulation-modeling framework.

HIGH-AUTONOMY CONCEPTS

Architecture

The definition of *intelligent autonomous system* as given by Erickson and Cheeseman (1986) stipulates the following behavior characteristics:

- The system must plan and replan to realize its goals.
- It must be able to execute its plans.
- It must monitor its environment.
- It must have cognitive capabilities.
- It must have diagnostic capabilities.

The behavior requirements determine a generic architecture for an autonomous system. This architecture is depicted in Fig. 1.

The flow of information and control in this architecture is defined as follows (Erickson and Cheeseman, 1986). The system's objectives (tasks, requirements,

performance measures, and constraints) are specified through the Interface Unit. The Planner defines a nominal plan to realize the system's objectives. The Simulator predicts the effects of the proposed plan based on its world model. The plan is interpreted by the Executor and carried out by way of the Effector unit. Changes to the state of the real system are detected by the Perceptor unit. These changes are observed by the Monitor, which compares them with the expected state of the real system based upon the results from the simulation-modeling process. If small adjustments are required, they can be directly passed on to the Executor. Otherwise, the Monitor passes information about large anomalies to the Diagnoser, which identifies the problem and sends a replaning request to the Planner. Central to the architecture is the CAD Methods Bank module. This module includes a system design methodology (the role of which we explain in detail in our section on support of design for autonomy) as well as the model and knowledge bases.

Antsaklis et al. (1989) refine the architecture by breaking up its functions into

1. Management and organization level, which determines the overall system's goals and supports interaction with the system's external environment through the interface unit;
2. Coordination level, which supports decision making, planning, and scheduling; it interfaces the management and execution levels;
3. Execution level, which carries out control actions determined at higher levels through automatic controllers and actuators; it collects data about the real world through sensors, monitors, and its perception system.

Furthermore, at each level there is a hierarchy of information flow that includes data, information, and knowledge (Zeigler, this issue).

Zeigler encapsulates knowledge in the form of models that can be employed at different levels of control in an autonomous system to support its objectives. The resulting structure is termed *model-based architecture.* The key distinguishing feature of Zeigler's approach is the use of partial models to deal with the multiplicity of the system's objectives and functions. Endomorphic modeling provides a promising framework to handle the complexity of autonomous systems design and simulation (Zeigler, 1990; Zeigler, this issue).

Degree of Autonomy

Achieving a certain degree of autonomy is an important requirement in design of intelligent autonomous systems (Antsaklis et al., 1989; Erickson and Cheeseman, 1986). One way to define the degree of autonomy is to consider the extent of a system's interaction with its environment through the interface unit.

The less the interaction (intervention of the human operator in the system's operation), the higher the system's autonomy (Erickson and Cheeseman, 1986). In its Telerobotics Project (Erickson and Cheeseman, 1986; NASA, 1985), NASA defines the degree of autonomy in terms of the level of detail and abstraction that the human operator has to employ when assigning tasks, and the length of time robots can function on their own without any intervention from their operator. In the model-based architecture, Zeigler (this issue) defines progressive levels of autonomy achievement as

- *Level 1:* The system should have the ability to achieve its objectives.
- *Level 2:* The system should be able to adapt to major environmental changes.
- *Level 3:* The system should be able to develop its own objectives.

In the ensuing sections we examine how a design methodology can support the development of systems capable of achieving high autonomy. When viewed in a design context, the Real World System component of the architecture in Fig. 1 is treated as a system being designed. Thus it is crucial that the approach we take to design, build, and eventually deploy this system meet the requirements for autonomy discussed heretofore.

Next, we discuss system design and describe a model-based framework called *knowledge-based simulation design methodology* (Rozenblit, 1985; Rozenblit and Zeigler, 1988, 1990). Then, we formulate postulates for applying this framework to high-autonomy system design.

SYSTEM DESIGN

We adopt the following definition: "Design is a goal directed activity producing a set of descriptions of an artifact that satisfy a set of given performance requirements and constraints" (Coyne, 1990). Design theory and methodology is an emerging area of research (Agogino et al., 1989; Bretschneider and Lagger, this issue, Coyne, 1990; Dixon et al., 1989; Maher et al., 1989; Rozenblit, 1985; Rozenblit and Huang, 1991; Rozenblit and Zeigler, 1988). It supports understanding of the design process and its automation in the form of computer-aided design. A number of methodologies and design systems have been developed to aid the engineering design process in different domains. Although different systems use different models to represent design knowledge and different systems work in different domains, some common features can be found and comparisons can be made among them.

A commonly taken view in knowledge-based, computer-aided design systems is that design is a search process in which a satisfactory design solution is produced from a number of alternatives (Coyne, 1990; Rozenblit, 1985; Rozenblit and Zeigler, 1991), which come from knowledge of the relevant domain.

The search proceeds in a design space that includes all knowledge and design decisions known so far. The design space can be expanded during the design process, that is, new knowledge and design decisions can be added to the space when the existing knowledge and available design decisions are not sufficient to obtain the design solution. If a design solution can be achieved using only existing knowledge in the design space, the process is then termed *routine design* (Coyne, 1990; Maher et al., 1989). In routine design, no new knowledge is added to the design space, and the design space is not expanded. If a design solution cannot be achieved using only the existing knowledge in the design space, new knowledge must be added. This kind of design is called *creative design* (Maher et al., 1989). In creative design, new knowledge is added into the design space, and thus the space is expanded. The new knowledge marks the creativity of design. The design, which is a search process in the design space, is guided by some principles. These principles are given by a design methodology.

An important feature of design methodologies is the representation method used in different stages of the design process. Some methodologies use a unique representation method throughout the whole design process, and some employ several different representation methods. Another characteristic is the *domain independence*. Some methodologies can be used only in a specific domain, while others are domain independent. Engineering design problems are often quite large and complex. The decomposition of the large, complex design problem into subproblems is another characteristic of design methodologies.

We have been developing a generic design methodology that embodies the specified characteristics (Rozenblit, 1985; Rozenblit and Huang, 1991; Rozenblit and Zeigler, 1988, 199). Its description follows.

Knowledge-based Simulation Design Methodology

The system design approach proposed by Rozenblit (1985) and by Rozenblit and Zeigler (1988, 1990), termed *knowledge-based simulation design* (KBSD), focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. It treats the design process as a series of activities that includes specification of design levels in a hierarchical manner (decomposition), classification of system components into different variants (specialization), selection of components from specializations and decompositions, development of design models, experimentation and evaluation via simulation, and choice of design solutions.

The design model construction process begins with developing a representation of design components and their variants. To appropriately represent the family of design configuration, we have proposed a representation scheme called the system entity structure (SES) (Zeigler, 1984a, 1990). The scheme captures

the decomposition, taxonomic, and coupling relationships among objects (entities) of a design domain.

Procedural knowledge is available in the form of production rules. They can be used to manipulate the elements in the design domain by appropriately selecting and synthesizing the domain's components. This selection and synthesis process is called *pruning* (Rozenblit and Huang, 1991; Rozenblit and Zeigler, 1988, 1990). Pruning results in a recommendation for a *model-composition tree,* that is, the set of hierarchically arranged entities corresponding to model components. A composition tree is generated from the system entity structure by selecting a unique entity for specializations and a unique aspect for an entity with several decompositions.

The final step in the framework is the evaluation of alternative designs. This is accomplished by simulation of models derived from the composition trees. Discrete event system specification (DEVS) (Zeigler, 1984a, 1984b, 1990) is a modeling formalism used for system specification in the methodology, providing a formal representation of discrete event systems. The performance of design models is evaluated through computer simulation, and alternative design models are evaluated with respect to experimental frames (Rozenblit, 1991; Zeigler, 1984a) that reflect design-performance questions. Results are compared and traded off in the presence of conflicting criteria. The consequence is a ranking of models and the support of choices of alternatives best satisfying the set of design objectives.

We now describe the SES representation, the pruning procedures for generating design configurations, and the simulation modeling layer of our design framework.

Design Model Structure Representation

As a step toward a complete knowledge-representation scheme for design support, we have combined the decomposition, taxonomic, and coupling relationships in a knowledge-representation scheme called the system entity structure (SES). Previous work (Rozenblit, 1985; Rozenblit and Huang, 1991; Rozenblit and Zeigler, 1988; Zeigler, 1984a, 1990) identified the need for representing the structure and behavior of systems in a declarative scheme related to frame-theoretic and object-based formalisms (Zeigler, 1990). The elements represented are motivated, on the one hand, by systems theory (Wymore, 1967; Zadeh and Desoer, 1963) concepts of decomposition (i.e., how system is hierarchically broken down into components) and coupling (i.e., how these components may be interconnected to reconstitute the original system). On the other hand, systems theory has not focused on taxonomic relations, as represented, for example, in frame-hierarchy knowledge-representation schemes. In

the SES scheme, such representation concerns the admissible variants of components in decompositions and the further specializations of such variants.

The interaction of decomposition, coupling, and taxonomic relations in an SES affords a compact specification of a family of models for a given domain. In an SES, *entities* are conceptual components of reality for which models may reside in a 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 possible variants of the entity.

The SES organizes possibilities for a variety of system decompositions and, consequently, a variety of model constructions. Its generative capability facilitates convenient definition and representation of models and their attributes at multiple levels of aggregation and abstraction. More complete discussions of the SES and its associated structure transformations are presented in Rozenblit (1985), Rozenblit and Huang (1991), Rozenblit and Zeigler (1988, 1990), and Zeigler (1990).

Rule-based SES Pruning

In the KBSD methodology, a model is synthesized from components stored in the model base. A synthesis specification is the result of pruning a substructure from the SES. Pruning can be viewed as a knowledge-based search through the space of candidate solutions to the design problem. Production rules are used to represent the knowledge consisting of modeling objectives, coupling constraints, users' requirements, and performance expectations. The aim of pruning is to recommend plausible model candidates for an optimal solution to the design problem (with respect to the design requirements and constraints).

The following steps are required to provide the rules that guide pruning of the SES: (1) for each specialization, specify a set of rules for selecting an entity; (2) for an entity with several aspects, specify rules for selecting a unique aspect; (3) for each aspect, specify rules that ensure that the entities selected from specializations are configurable, that is, the components they represent can be validly coupled. Thus we have two types of rules:

Selection Rule Set: Selection Rules are associated with entities that have children (aspects or specializations). The rules determine which specialization or aspect should be selected.

Synthesis Rule Set: Synthesis rules are associated with aspects. The rules check whether the aspect's entities are configurable with respect to the coupling constraints.

Each rule can be assigned a certainty factor indicating the rule's degree of applicability.

These rule sets constitute a knowledge base for the inference engine that prunes an SES for a particular application domain. Pruning results in a recommendation for a *model-composition tree*.

To support the model-construction process, we have available a set of software tools that are currently being integrated on AI workstations and PC's. An expert system shell, MODSYN (*mod*el *syn*thesizer) (Rozenblit and Huang, 1991), was developed and implemented to generate model structures. MODSYN uses selection and synthesis rules to generate a recommendation for a composition tree. A suitable simulation environment can, given a composition tree, automatically generate a model ready to simulate.

Modeling and Simulation

DEVS-Scheme is a simulation environment that synthesizes executable simulation models from a composition-tree specification (Zeigler, 1984a, 1984b, 1990). It thus serves as the modeling and simulation layer underpinning the KBSD methodology. The DEVS formalism introduced by Zeigler (1984a, 1990) provides a means of specifying a mathematical object called a *system*. Basically, a system has a time base, inputs, states, and outputs, as well as functions for determining the next states and outputs when current states and inputs are given. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. DEVS-Scheme, an implementation of the DEVS formalism in Scheme (a Lisp dialect), supports building models in a hierarchical, modular manner. This is a systems-oriented approach not possible in popular commercial simulation languages. More detail regarding the formalism and simulation environment is available in Zeigler (1990).

Experimental Frame Specification

Rozenblit (1991) has developed a methodology for defining conditions under which simulation models can be observed and experimented with. Such conditions are formalized as *experimental frames*, that is, specifications of circumstances in which a model (or a real system) is observed and experimented with (Rozenblit, 1991; Zeigler, 1984a, 1990). An experimental frame reflects modeling objectives since (1) it subjects a model to input stimuli (which represent potential interventions into the model's operation); (2) it observes the model's reactions to the input stimuli and collects the data about such reactions (output data); and (3) it controls the experimentation by placing relevant constraints on

values of the designated model state variables and by monitoring these constraints.

Experimental frames are given concrete form. Employing the concepts of automata theory and the DEVS formalism, Zeigler (1984a) defines a *generator,* which produces the input segments sent to a model, an *acceptor,* a device that continually tests the run-control segments for satisfaction of the given constraints, and a *transducer,* which collects the input/output data and computes the summary mappings.

The experimental frame specification methodology defines two schemes for carrying out simulation experiments with hierarchical, modular models: (1) the centralized architecture is based on a global experimental frame that specifies conditions for the entire model; (2) the distributed architecture facilitates attachments of frame components to model simulators at different levels of the model hierarchy.

Phases in the KBSD Methodology

The phases required to execute the KBSD methodology are:

1. Decompositions and specializations of components of the system being designed are conceptualized using the SES. We utilize the SES base as a repository of previous design-modeling experience. Thus, we may retrieve an entity structure from this base that is applicable to the modeling domain at hand. Such an entity structure is modified and enhanced with entities required in the new project. Models associated with new atomic entities must be developed and placed in the model base.
2. A rule base to be used in the pruning process is developed.
3. The pruning engine is invoked, generating recommendations for candidate solutions to the design problem in the form of model-composition trees.
4. The transformation procedures that synthesize design models from the composition trees obtained in phase 3 are executed.
5. Relevant experimental frames that reflect design objectives are defined.
6. Simulation results are evaluated, and design models are ranked.

These phases may be iterated in a feedback process, as depicted in Fig. 2.

In next section, we explore how the KBSD methodology can support high-autonomy systems design.

SUPPORT OF DESIGN FOR AUTONOMY

Central to the architecture of Fig. 1 is the module CAD Methods Bank, which integrates the other components of the architecture. We propose that it

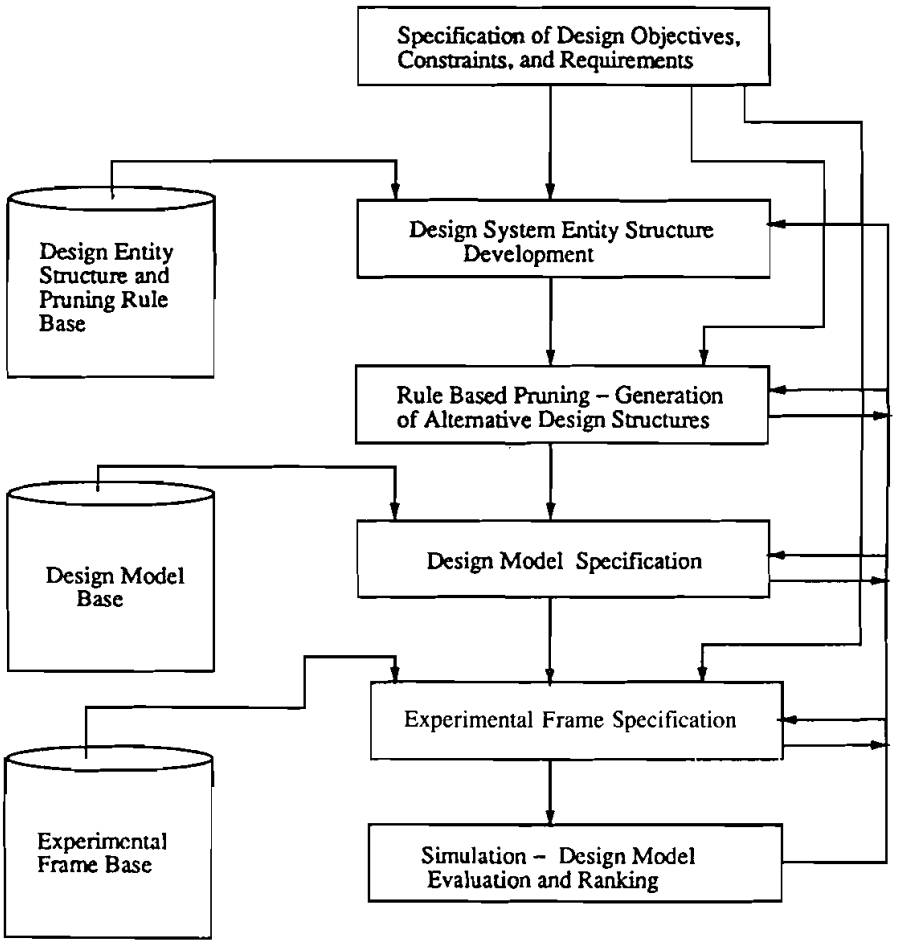

FIGURE 2. Knowledge-based simulation design framework.

contain a methodology capable of supporting design, development, deployment, evaluation, redesign, and maintenance of an autonomous system. We stipulate here that the real-world component (the real system) be designed concurrently with the other fundamental modules of the architecture of Fig. 1. We believe that taking this approach will reduce the complexity of the management, coordination, and execution layers of the architecture and will increase the degree of autonomy of the complete design.

Consider, for example, design of flexible manufacturing systems. In recent years, machining and assembly of products has moved toward a partial or complete automation, which will require higher autonomy. Extensive efforts are under way to improve the efficiency and cost effectiveness of manufacturing systems (Kusiak, 1990; Lenz, 1989).

To produce an artifact or carry out a task (machining, assembly, testing, etc.), a production plan is developed that organizes a set of technological operations into a sequence of actions leading to the final product (Jacak and Rozenblit, 1991a, 1991b; Kusiak, 1990; Rozenblit and Jacak, 1991). These actions, carried out by robots and automated devices, take place inside a production plant whose physical layout greatly influences the performance of the overall system. The physical layout of components on the work scene is the basis for robot-motion planning, along with the kinematic and dynamic characteristics of the robot itself. In this perspective, spatial design is the key component in realizing a production plan and in achieving high efficiency of the system's operation. The layout should be designed by taking into account not only the physical constraints but also the assembly production sequence. Hence, the interplay of behavioral and structural constraints such as material flow among devices, operations' precedence relations, safety or ease of maintenance, etc., is essential in arriving at an efficient design. We elaborate this point further in the section on autonomous system synthesis.

We now proceed to examine how the tools and methods of the KBSD methodology can support design for high autonomy.

Design-Space Structuring

We postulate that the SES must be used to represent knowledge about the components of a design domain. The SES representation must include the Real World/System and Autonomous Architecture entities depicted in Fig. 3.

The Autonomous Architecture has a decomposition that reflects the requirements previously defined in the section on system design. Thus, at the highest level of abstraction, all the major components (i.e., Perceptor, Effector, Executor, etc.) must be identified in the SES. These components can be further decomposed and classified, resulting in a knowledge base of components that can be used in domain-specific design. To illustrate this point, let us consider the Perceptor module. As shown in Fig. 3, this module may have a wide range of sensors, for example, temperature and pressure sensors, tactile sensors, range detectors, television cameras, audio sensors. Similarly, the Effectors may include activating relays, servomotors, valve shutoffs, etc. Requirements and constraints for a specific design problem will determine which sensors and actuators should be used in the system being designed. For example, high-precision tactile sensors will not be recommended for applications in which objects operate in extremely high temperatures. We shall return to the issue of component selection shortly.

In addition to the structure representation of the Autonomous Architecture, its operational aspects can be captured by an SES as well. Zeigler and Chi

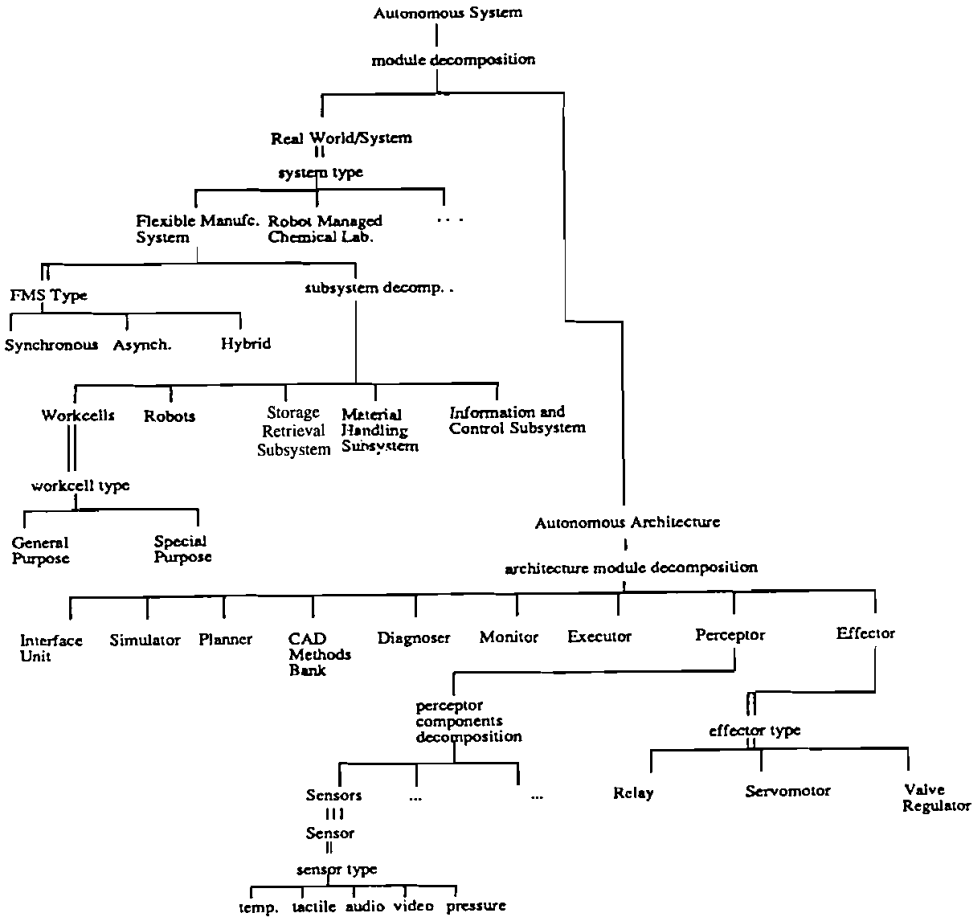

FIGURE 3. High-level SES of an autonomous system.

(1990) and Luh and Zeigler (1991) give several examples that illustrate SES-based plan generation for a robot-managed chemical laboratory.

The Real World/System component and its design have not received adequate attention in the literature that reviews autonomous systems. We stress again that the Real World/System design process should be an inherent phase in the development of a complete, autonomous system. We must have methods available to establish how the environment in which the autonomous architecture is to operate affects the design of this architecture. Conversely, the autonomy requirements will also impinge on the Real World/System design. The entity structure should be used to represent the Real World/System's design domain, its components, their attributes, decompositions, and taxonomies. Recall Fig. 3, where the Real World/System entity may be a flexible manufacturing system.

Such a system can have various functional types (synchronous, asynchronous, or hybrid) and several different subsystems. Selection of specific components and configuration of the final system will depend in great measure on the constraints and requirements needed to achieve autonomy. We discuss this issue next.

Autonomous System Synthesis

In this phase, a design description (in terms of the system's structure and topology) is generated. In our design framework, this is accomplished by rule-based pruning (Rozenblit and Huang, 1991).

The definition of rules that will select and synthesize specific components of the real system and the autonomous architecture layers poses a very difficult task. The difficulty lies mainly in translating the autonomy requirements into design constraints. The general desiderata (e.g., the system must plan to realize its goals; it must monitor its environment; it must have diagnostic capabilities) translate into selection of planners, monitors, diagnosers, etc. However, the definition of what constitutes a high degree of autonomy is still imprecise. Thus, the formulation of design constraints is largely dependent on what designers perceive as desirable characteristics, and it is done on a case-by-case basis (Erickson and Cheeseman, 1986). For example, many designers consider a mobile robot to be more autonomous than a fixed one. When synthesizing a robot-based automation system, the higher the autonomy required, the more likely it is that the designers will select mobile robots.

The degree of control exerted by a designer (and eventually the operator) over the autonomous system's environment is also an important factor. If this degree of control is high, then the role of the autonomous system layer in the overall design is to monitor and check to be sure that the operational objectives are met. This represents level 1 autonomy, as defined in the section on degree of autonomy. If the degree of control is low, then the autonomous architecture needs high autonomy. For example, in the manufacturing-systems domain, the level of autonomy required usually falls in between. Typically, diagnostics is a difficult issue, and it is desirable that the problems be identified automatically. However, once the failure and its reasons have been identified, repairs can be performed by human operators.

The engineering-type constraints are easier to handle. For example, if device-temperature measurement is required, we must select a temperature sensor or if an assembly system is to operate in hazardous conditions (e.g., high toxicity), only robot-managed workcells can be used.

To summarize, we believe that the crucial issues here are (1) establishment of a tangible representation of degree of autonomy, (2) translation of this representation into rules for selection and configuration of the autonomous architecture and its environment, and (3) development of pruning mechanisms that will

enable us to interface planning of the execution sequence with planning of the real system structural design. We have pointed out the importance of this problem in the context of manufacturing systems (layout design) (Jacak and Rozenblit, 1991a, 1991b; Rozenblit and Jacak, 1991).

Modeling

The key supposition of our approach is the use of simulation models to evaluate alternative design solutions. The SES/model-based framework proposed by Zeigler (1990) is employed to generate models of the real system, to generate families of planning alternatives, and to build a hierarchical event-based control structure. We refer the reader to Zeigler's paper in this issue for further details.

Execution and Performance Evaluation

In principle, an autonomous system could base its operation on a comprehensive model of its environment (and itself). However, to develop such a model would be an intractable task. Instead, in the model-based architecture, partial models of different levels of abstraction are employed. The partial models are oriented toward specific objectives, and thus they need to be evaluated in respective experimental frames that reflect those objectives.

The experimental frame-specification methodology (Rozenblit, 1991) integrates well with the model-based approach to high-autonomy systems. First, it provides a systematic approach to defining a set of conditions under which an autonomous system is to operate and to measuring the degree to which the system is capable of performing certain actions. The system's ability to achieve a prespecified objective can be tested within a frame that defines this objective. The ability to adapt to major environmental changes can be measured by emulating the changes through an appropriate experimental frame. Testing the ability of the system to develop its own objectives will involve not only the creation of new models to support the new objectives but also the development of relevant experimental frames that reflect those objectives.

Second, the distributed frame architecture (Rozenblit, 1991) supports flexible experimentation with multicomponent systems that may exhibit various degrees of distribution and coordination among their components. It is the level of abstraction at which we wish to observe the behavior of the system that determines where we attach the frame components and how we define their functions. Thus, the degree of autonomy of individual system components may be observed in local frames that pertain to those components or within higher-level frames that assess the coordination and cooperation among the components.

CONCLUSION: ACHIEVING AUTONOMY THROUGH DESIGN

To conclude, we assess the value of a general methodology to support design of high-autonomy systems. We first revisit the design phases, assign design responsibilities, and point to critical issues.

- *Design-Space Structuring:* Domain experts assist in the development of the SES. This is a multidisciplinary effort, involving experts in application domain, AI, control, and modeling.
  Problems: None are significant because most engineering applications are well structured. A repository of knowledge from previous designs is almost always available, and new designs are rarely entirely innovative.
- *Autonomous System Synthesis:* Domain experts translate constraints into pruning rules.
  Problems: The notion of autonomy is intangible; autonomy definition in terms of performance indexes required by the design methodology is lacking; no procedures exist for amalgamating execution plans into the structural system design.
- *Modeling:* Simulation experts are responsible for establishing the model base. Problem: It is difficult to establish adequate modeling abstractions.
- *Performance Evaluation:* Simulation experts are responsible for establishing the adequate experimental frame base.
  Problem: It is difficult to achieve simulation efficiency in real-time applications.

We believe that a general methodology such as KBSD unifies the activities necessary to accomplish an autonomous system design. Several advantages of using the methodology are apparent:

1. The SES and pruning algorithm facilitates rapid knowledge-based selection and configuration of components in the design domain. Structure reconfiguration, which may become necessary due to major changes in environment or failures of equipment, can be enabled by repruning the SES underlying the design at hand.
2. Designs are modeled and simulated prior to being deployed. This considerably reduces the cost of system implementation and its potential redesign.
3. Endomorphic modeling facilitates operational management of the autonomous architecture, that is, planning, scheduling, diagnosis, and control.
4. Experimental frames are a means of collecting quantitative data about the degree to which a system is capable of achieving autonomy.
5. The methodology can integrate design of an autonomous architecture with the environment in which the architecture is to be deployed.

Implementing our methodology in a specific, autonomous system design context will no doubt require a considerable effort, especially in integrating all the design phases. Our current work focuses on computer-aided design of flexible manufacturing systems (Jacak and Rozenblit, 1991a, 1991b; Rozenblit and Jacak, 1991).

REFERENCES

Agogino A. M., Bradley, S. R., Cagan, J., Pramod, J., and Michelena, N. 1989. AI/OR Computational model for integrating qualitative and quantitative design methods. *Proc. NSF Engineering Design Research Conference,* pp. 97–112. Amherst, Mass.

Antsaklis, P. J., Pasino, K. M., and Wang, S. J. 1989. Towards intelligent autonomous control systems: Architecture and fundamental issues. *J. Intel. Robot. Syst.* 1(4):315–342.

Coyne, R. D. 1990. *Knowledge-based Design Systems.* Reading, Mass.: Addison-Wesley.

Dixon, J. R., Guenette, M. J., Irani, R. K., Nielesen, E. H., Orelup, M. F., and Welch, R. V. 1989. Computer-based models of design processes: The evaluation of designs for redesign, *Proc. NSF Engineering Design Research Conference,* pp. 491–506, Amherst, Mass.

Erickson, W. K., and Cheeseman, P. C. 1986. Issues in design of an executive controller shell for space station automation. *Opt. Eng.* 25(11):1194–1199.

Fishwick, P. A., Rozenblit, J. W., and Zeigler, B. P., eds. 1991. *Proc. 2d Conference on AI, Simulation, and Planning in High Autonomy Systems.* IEEE Press.

Jacak, W., and Rozenblit, J. W. 1991a. Automatic robot programming in CAST. In *Lecture Notes in Computer Science,* ed. F. Pichler, New York: Springer-Verlag.

Jacak, W., and Rozenblit J. W. 1991b. Automatic simulation, interpretation and testing of a task-oriented robot program for a sequential technological process, *Robotica.*

Kusiak, A. 1990. *Intelligent Manufacturing Systems.* Englewood Cliffs, N.J.: Prentice Hall.

Lenz, J. E. 1989. *Flexible Manufacturing.* New York: Dekker.

Luh, C. J., and Zeigler, B. P. 1991. Abstraction morphisms for task planning and execution. *Proc. of the 2d Conference on AI, Simulation, and Planning in High Autonomy Systems,* pp. 50–59. IEEE Press.

Maher, M. L., Zhao, F., and Gero, J. S., 1989. An approach to knowledge-based creative design. *Proc. NSF Engineering Design Research Conference,* pp. 333–346, Amherst, Mass.

NASA. 1985. The Space Station Program. NASA Publication.

Rozenblit, J. W. 1985. A Conceptual Basis for Integrated, Model-based System Design. Unpublished Ph.D. thesis, Department of Computer Science, Wayne State University, Detroit.

Rozenblit, J. W. 1991. Experimental frame specification methodology for hierarchical simulation modeling. *Int. J. Gen. Syst.* 19(3):317–336.

Rozenblit, J. W., and Huang, Y. M. 1991. Rule-based generation of model structures in multifacetted modeling and system design. *ORSA J. Comput.* 3(4):330–344.

Rozenblit, J. W., and Jacak, W. 1991. Simulation based planning of robot tasks in flexible manufacturing, *Proc. 2d conference on AI, Simulation and Planning in High Autonomy Systems,* pp. 166–173. IEEE Press.

Rozenblit, J. W., and Zeigler, B. P. 1988. Design and modeling concepts. In *International Encyclopedia of Robotics, Applications and Automation,* ed. R. Drof, pp. 308–322. New York: Wiley.

Rozenblit, J. W., and Zeigler, B. P. 1990. Knowledge-based simulation design methodology: A flexible test architecture application. *Trans. Soc. Comput. Simul.* 7(3):195–228.

Rozenblit, J. W., and Zeigler, B. P. 1991. *Proc. 1st Conference on AI, Simulation, and Planning in High Autonomy Systems.* IEEE Press.

Sardis, G. N. 1983. Intelligent robotic controls. *IEEE Trans. Automa. Contr.* AC-28(5):547–556.

Wymore, A. W. 1967. *A Mathematical Theory of Systems Engineering: The Elements.* New York: John Wiley.

Yang, J., and Rozenblit, J. W. 1990. Case studies of design methodologies: A survey. *Proc. International Conference on AI, Simulation and Planning in High Autonomy Systems,* pp. 136–141, IEEE Computer Press.

Zadeh, L. A., and Desoer, C. A. 1963. *Linear Systems Theory: The State Space Approach*. New York McGraw-Hill.

Zeigler, B. P. 1984a. *Multifacetted Modelling and Discrete Event Simulation*. London: Academic Press.

Zeigler, B. P. 1984b. System-theoretic representation of simulation models. *IIE Trans*. March: 19–34.

Zeigler, B. P. 1990. *Object-oriented Simulation with Hierarchical Modular Models; Intelligent Agents and Endomorphic Systems*. San Diego: Academic Press.

Zeigler, B. P., and Chi, S. D. 1990. Model-based architectures for autonomous systems. *Proc. 1990 IEEE Symposium on Intelligent Control*. Philadelphia. pp. 27–32, Philadelphia.