Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12,* 193–209. Printed in the USA.

A foundation for machine learning in design

SIANG KOK SIM[1] AND ALEX H.B. DUFFY[2]

[1]School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Republic of Singapore
[2]CAD Centre, University of Strathclyde, 75 Montrose Street, Glasgow G1 1XJ, Scotland, UK

(RECEIVED June 27, 1997; REVISED October 17, 1997; ACCEPTED November 10, 1997)

Abstract

This paper presents a formalism for considering the issues of learning in design. A foundation for machine learning in design (MLinD) is defined so as to provide answers to basic questions on learning in design, such as, "What types of knowledge can be learnt?", "How does learning occur?", and "When does learning occur?". Five main elements of MLinD are presented as the input knowledge, knowledge transformers, output knowledge, goals/reasons for learning, and learning triggers. Using this foundation, published systems in MLinD were reviewed. The systematic review presents a basis for validating the presented foundation. The paper concludes that there is considerable work to be carried out in order to fully formalize the foundation of MLinD.

Keywords: Design Knowledge; Design Process Knowledge; Design Reuse; Knowledge Transformation/Change; Learning in Design; Machine Learning Techniques

1. INTRODUCTION

Design experience provides a wealth of knowledge that designers can (re)-use to design better products within a shorter time-to-market period and at the same time be economically competitive. The research work in the area of machine learning has contributed many methods that have been applied to the acquisition of knowledge in design. This has been evident from the body of work reported in the field of machine learning in design (MLinD) (Duffy, 1997). What is evident from this work is the application of particular machine learning methods to the acquisition of some specific design knowledge. In the area of MLinD, key questions raised by Persidis and Duffy (1991) are now being addressed (Duffy, 1997):

- *What* type of knowledge is learned?
- *How* is learning taking place?
- *When* is learning taking place?

Given that there is sufficient evidence from MLinD research in which generalized past design knowledge can be acquired using certain machine learning methods, there is now a need for a systematic approach to formalizing learning in design. Leith (1990) argues strongly for a formalism to arrive at the state of what "ought to be" in artificial intelligence (AI) and computer science rather than what "is" (i.e., current state of these disciplines) so as to overcome the *ad hoc* basis of software writing and the inefficient development of software systems. Using the same argument of Leith, given that machine learning in design is a specialized application of AI and computer science, there is therefore a need to put the study of MLinD research and the development of the MLinD systems to support designers on a formal basis.

To answer the questions raised in a structured basis, Section 2 presents basic elements of learning and a foundation for learning using these elements. This foundation is presented here as a basis for the "dimensions of machine learning" raised by Grecu and Brown (1996) and similarly by Persidis and Duffy (1991). Since the research work in MLinD has resulted in the development of many published systems in support of knowledge acquisition in design, this foundation has been used to analyze and evaluate these MLinD systems. The five elements presented here are the input knowledge, the output knowledge, the knowledge transformer, the learning triggers, and the learning goal. Section 3 presents the types of design knowledge learned (both

Reprint requests to: Dr. A.H.B. Duffy, CAD Centre, University of Strathclyde, 75 Montrose Street, Glasgow G1 1XJ, Scotland, UK. Tel: (+44)141-548-3134; Fax: (+44)141-552-3148; E-mail: alex@cad.strath.ac.uk

https://doi.org/10.1017/S0890060498122096 Published online by Cambridge University Press

product knowledge and design process knowledge) given the input knowledge and the goal of learning as described in the MLinD systems. Section 4 provides evidence of various knowledge transformers used to learn design knowledge but implemented in various machine learning methods and techniques. Evidence of the types of learning triggers and when learning occurs are described in Section 5. Section 6 compares the foundation for machine learning with "dimensions of machine learning" raised by Grecu and Brown (1996) and concludes that the paper presents a structured basis upon which to research and develop the field of machine learning in design.

2. DEFINING A FOUNDATION OF DESIGN LEARNING

It is proposed that a systematic approach to the study of learning design knowledge can be based on analyzing the knowledge change of design activities. This perspective is based upon the hypothesis posited by Persidis and Duffy that learning is inextricably linked to design. In this paper, learning in design is viewed as a knowledge-gaining activity associated with the activity of design.

A foundation for learning in design must be able to address some of the key questions in learning that were raised in Section 1. Since learning can be viewed as an activity, any formalism must invariably consider what knowledge is input into that activity, what is the output knowledge, and the knowledge change that transforms the input knowledge into output knowledge. Knowledge change involves the transformation of the existing knowledge into some new knowledge. Since there are many possible ways in which knowledge can be transformed, it is necessary to define what kind of knowledge can be learned. For example, given a past design, one can learn about the composition of the product in terms of "part-of" hierarchy, or about the relationships between attributes of components. Further, the learning activity often will have a specific goal, for example, gain new knowledge of a product, explore and generalize a particular domain. It is also necessary to define what triggers learning and when that trigger can occur.

$$\begin{tikzcd}
 & G_l \arrow[d] & \\
I_k \arrow[r] & K_t \arrow[r] & O_k \arrow[ul] \arrow[dl] \arrow[ll, bend right] \\
 & T_l \arrow[u] &
\end{tikzcd}$$

Fig. 1. Elements of learning.

Given the above, the basic elements of a learning activity may consist of:

- existing knowledge as input knowledge, I_k;
- knowledge transformers, K_t;
- output knowledge, O_k;
- learning goal or reason, G_l;
- learning triggers, T_l (T_{lw} and T_{lt}).

These basic elements of learning may be related as shown in Figure 1.

In this figure, it is suggested that the input knowledge is transformed into new output knowledge, which then can feed back into the learning activity as input knowledge for yet more new knowledge. This output knowledge in itself may also trigger or act as a reason or goal for a learning activity.

2.1. Elements of a foundation for machine learning in design

Using the above elements as a basis, we now can map Persidis and Duffy's, and Grecu and Brown's issues and "dimensions" as shown in Table 1. It can be noted from Table 1 that Grecu and Brown go into far greater detail than Persidis and Duffy, possibly reflecting the evolution of our

Table 1. *Elements of machine learning in design*

Basic elements of learning	Persidis and Duffy	Grecu and Brown
I_k (input knowledge)	• Not explicitly addressed	• What are the elements supporting learning? • Availability of knowledge for learning
K_t (knowledge transformer)	• How is learning carried out?	• Methods of learning • Local vs. global learning
O_k (output knowledge)	• What knowledge is learned?	• What might be learned?
T_l (trigger)	• What can trigger learning? • When is learning triggered?	• What can trigger learning?
G_l (goal/reason)	• Not explicitly addressed	• Consequences of learning

own understanding of the field. Having said this, the approaches seem complementary and would seem to reflect the basic elements identified above. Some issues associated with the above are discussed further.

2.2. Types of design knowledge learned

The types of design knowledge to be learned are dependent on the activity of the design process, the types of input knowledge, the goal of the learning process, and when learning takes place. Persidis and Duffy (1991) state five main types of design-related knowledge: design requirements, design description, domain knowledge, case histories, and design management. Failures in design provide an opportunity to learn about the causes of the failure in order to avoid committing the same failure again. Knowledge learned from design failure could be in the form of types of failures and conflicts (Grecu & Brown, 1996; Vancza, 1991), heuristics for failure recovery, and conflict resolution (Grecu & Brown, 1996; Vancza, 1991). On the other hand, successful designs and design processes (Grecu & Brown, 1996; Vancza, 1991) also provide opportunities for learning about the characteristics of the successful past designs and goals/plans of design processes. The review of MLinD systems in Section 3 shows that indeed there are numerous types of design knowledge that can be learned from past designs. It is therefore reasonable to conclude that there are many types of design knowledge that can be learned.

2.3. The learning goal

Because of the variety of knowledge that can be learned, it is sometimes necessary to specify a learning goal. The learning goal influences what parts of the existing knowledge are relevant, what knowledge is to be acquired, in what form, and how the learned knowledge is to be evaluated. Such goals, as in design, can be hierarchical in nature in that a specific goal could be a subset of a more general goal. For example, learning about a particular technology may help improve the quality of the product or speed up the design process. Another example of a specific learning goal such as learning about constraint violation in design may result in better conflict anticipation and resolution, which in turn may lead to a better design plan and hence a shorter design cycle (Leith, 1990). Thus, specific learning goals may be the stimulus for many of the unforeseen consequences of learning.

2.4. Types of knowledge change

The design process is the vehicle that designers use to change the state of the design through the application of scientific and design knowledge. As the state of the design changes, there is a corresponding change of knowledge state of the design. Persidis and Duffy presented three main classifications under which knowledge change can be considered, namely, acquisition, modification, and generation:

- *Acquisition*: the process in which new knowledge, either through direct input from the user or derived from knowledge sources, is received.
- *Modification*: the process of altering the existing structure of knowledge through either addition or deletion of the knowledge component from the structure. The modified knowledge structure remains the same type.
- *Generation*: the process of creating completely new knowledge from existing knowledge.

While knowledge change through modification is explicitly described as the addition/deletion of knowledge component from the knowledge structure, the several ways in which knowledge change through generation is achieved are described below. The different processes by which knowledge change takes place are called *knowledge transformers.* In the study of machine learning in design systems, seven knowledge transformers for knowledge generation and one knowledge transformer for knowledge modification are identified. Some of these transformers are similar to those described by Michalski's set of 11 knowledge transmutations (Michalski, 1993). Of these 11, the knowledge transmutation of replication/destruction, sorting/unsorting, and selection/generation are not considered here as knowledge transformers because they are basically processes for reorganizing knowledge bases. There is no learning in the form of knowledge change *per se.*

2.4.1. Knowledge transformers

Because of the nature of design knowledge, which may exist in symbolic or numerical form, knowledge change operators or knowledge transformers consist of two types: the symbolic type and/or the non/subsymbolic type. A knowledge transformer is an operator that derives a piece of new knowledge from a given input or an existing piece of knowledge.

Group rationalization (or clustering)/decomposition (ungroup). Group rationalization involves the grouping of past designs according to their similarities when considering particular perspective(s) or criteria (Duffy & Kerr, 1993). The grouping may be based on single attributes and/or nested attributes. Decomposition removes the groupings.

Similarity/dissimilarity comparison. Similarity comparison derives new knowledge about a design on the basis of similarity between the design and similar past design(s). The similarity comparison is based on analogical inference. Case-based design is a specific application of learning by similarity comparison in which the source of past design knowledge is a set of similar design cases rather than generalized design rules. The opposite is dissimilarity comparison, which derives new knowledge on the basis of the lack of similarity between two or more past designs.

Association/disassociation. Association determines a dependency between given entities or descriptions based on some logical, causal, or statistical relationships. For example, two entities may be related together by a taxonomic relationship (i.e., "kind-of") or compositional relationship (i.e., "part-of"). The opposite is disassociation, which asserts a lack of dependency (e.g., "not part-of").

Derivations (reformulation)/randomization. Derivations are transformations that derive one piece of knowledge from another piece of knowledge (based on some dependency between them). In contrast, randomization transforms one knowledge segment into another by making random changes. Because the dependency between knowledge components can range from logical equivalence to a random relationship, derivations can be classified on the strength of dependency.

Generalization/specialization. Generalization generates a description that characterizes all of the designs based on a conjunction of all of the specializations of that concept. Typically, the underlying inference is inductive. But generalization also can be deductive, when the more general description is deductively derived from the more specific one using domain knowledge or existing knowledge (e.g., as in explanation-based generalization). Specialization increases the specificity of the description.

Abstraction/detailing. Abstraction generates a new version of the knowledge with less detail than the original through the use of representation of abstract concepts or operators. As distinct from generalization, the underlying inference is deductive in nature. Detailing is the opposite, in which the new knowledge is generated with more details.

Explanation/discovery. Explanation derives additional knowledge based on domain knowledge. Discovery derives new knowledge without an underlying domain knowledge.

The knowledge transformers described above are not synonymous with the machine learning methods that are used in machine learning systems. The knowledge transformers describe the cognitive learning process whereas the machine learning techniques describe how that process can be implemented as algorithms. For example, explanation-based generalization uses generalization and explanation of domain knowledge to derive new knowledge.

2.5. Learning triggers

Persidis and Duffy (1991) posit the need to identify events that act as "triggers" for the self-activation of the learning process. In this paper, it is suggested that these triggers can be classified into three main categories: in situ, provisional, and retrospective.

- *In situ triggers* are activated when there is a need to acquire new knowledge while the design is under focus of attention.
- *Provisional triggers* are activated when there is a foreseen event that is envisaged to require additional knowledge. An example of a provisional trigger arises when there is a need to learn heuristics that may lead to a reduced search for a design solution. This learning is to be done in anticipation of or provisionally for a foreseen event.
- *Retrospective triggers* are activated after an event. That is, learning is triggered by the need to learn from successful design(s)/failed design(s) and/or processes in hindsight.

Grecu and Brown (1996) identify several situations that could trigger learning. The reasons for learning are numerous but may occur due to the following:

- *Novelty driven*: when there is a new design problem, a new technology, a new process, or a new design requirement.
- *Excellence driven*: can the design be improved to achieve better quality and reliability? Can the design process be more streamlined or time-efficient or resource-efficient? Can the design be market-competitive?
- *Failure avoidance driven*: can failure(s) in the design be avoided? Can design constraints be overcome?

The above examples serve to illustrate reasons that give rise to learning and are by no means exhaustive in coverage. For specific examples (gleaned from the study of MLinD systems) of what triggers learning and when learning is triggered, more details are given in Section 5 and Table 3.

2.6. Current work of machine learning in design

There is now considerable effort in supporting computer-based learning in design. Such work can act as a basis upon which to discuss the above elements. The examination of such systems can be broadly classified into two groups:

- systems that acquire knowledge of a *product* (i.e., design concepts);
- systems that acquire knowledge of the *design process* (i.e., procedural knowledge and control knowledge).

It is difficult to ascertain the true nature of the goal or reason for an associated MLinD system's learning activity. Consequently, the examination has concentrated on attempting to identify:

- the types of design knowledge acquired (Element O_k) and the associated learning goal (Element G_l);
- the knowledge change as a result of the learning activity (Elements O_k/I_k);
- the knowledge transformers (Element K_t) and the corresponding machine learning method(s) used in implementation;

- what triggers that learning activity and when (Element T_l).

3. TYPES OF DESIGN KNOWLEDGE LEARNED

Many MLinD systems have been developed to automate the learning of design knowledge. The purpose of this section is to show the variety of design knowledge that has been acquired through these MLinD systems. For the convenience of description, the types of design knowledge learned can be categorized into the following:

- product design knowledge;
- design process knowledge.

This categorization is important because it was revealed through the examination of the MLinD systems that, in most cases, the learning of meaningful product design knowledge occurs retrospectively. Similarly, certain types of design knowledge are best learned while the design is in progress.

3.1. Product design knowledge

Systems such as BRIDGER (Reich, 1993), NODES (Persidis & Duffy, 1991; Duffy et al., 1995), CONCEPTOR (Li, 1994), PERSPECT (Duffy & Duffy, 1996), and NETSYN (Ivezic & Garrett, 1994) have addressed the issue of learning product knowledge from past design concepts. The knowledge learned can be discussed under the headings of:

- composition of the components/subsystems that constitute the product;
- constraints imposed on the attributes due to the physics of the problem, geometrical and spatial requirements, life-cycle issues, design requirements, and so on;
- decomposition of the components that constitute the product;
- performance evaluation knowledge;
- dynamic learning of implicit knowledge.

3.1.1. Learning knowledge of design composition

Knowledge of the composition of past designs serves as as a useful starting point to initiate the synthesis process of design problems that have a similar design specification. For example, BRIDGER was developed primarily for the synthesis of different concepts of cable-stayed bridges. NODES also was developed primarily to support concept modeling operations during the early stages of design (Duffy et al., 1995).

While BRIDGER expresses the design composition knowledge as a taxonomic hierarchy (i.e., "kind-of" relationship), NODES expresses both the taxonomic relationships as well as the compositional ("part-of") relationships of a design object. BRIDGER uses the concept formation capability of ECOBWEB to learn design composition knowledge, and NODES uses maximum conjunctive generalization (MCG) to build the generalization of a new concept into existing concept(s).

The knowledge input into NODES or BRIDGER consists of past design instances or abstractions in terms of taxonomic hierarchy (i.e., "kind-of" relationship and/or "part-of" relationship). Through the process of knowledge change (i.e., clustering or generalization) the output knowledge will be a new taxonomic structure updated to reflect the changes caused by the inclusion of the new design instance.

3.1.2. Learning empirical knowledge of quantitative and qualitative relationships

The numerical relationships of attributes of design concepts are important in defining the preliminary definition of the form or structure of the concept. The estimation of these values has been achieved by scaling or interpolation (BRIDGER; Reich, 1993), empirical network of association (CONCEPTOR; Li, 1994), statistical approximation (NODES using Designer; Duffy et al., 1995), and neural networks (NETSYN, Ivezic & Garrett, 1994). Each of these types of estimation of attribute values based on records of past design cases is described below.

Although there is in BRIDGER a taxonomic structure generated from past design concepts in terms of attribute value pairs, the numerical relationships of these attributes are not explicitly represented. Therefore, for a given design specification, BRIDGER derives appropriate values for the attributes through the interpolation of suitable design cases.

In the case of CONCEPTOR, the empirical relationships of design attributes of a given design domain are represented as an empirical network. The empirical network captures both quantitative variables and qualitative variables as nodes, which can be linked by empirical formulas (for quantitative variables) or design patterns (for qualitative variables). The empirical formula represents proportional relationships among the quantitative variables. Design patterns capture important associations among qualitative variables on which the designer should focus his or her attention. Each node represents a design feature, its default value, and range value. Collectively, the empirical network of nodes linked either by empirical formula or design patterns represents the description attributes of a learned concept.

In NODES, the numerical relationship of attributes of the design object is described by characteristics and formulas. Characteristics represent numerical attributes of concepts and always are associated with a single object. They also are associated with formulas and together form a constraint network where the values of characteristics are constrained by the values of other characteristics appearing in the same formula.

Formulas represent mathematical equations that allow the values of characteristics to be calculated from other characteristics. Formulas, like characteristics, also are associated with objects, but, unlike characteristics, they can be associated with more than one object.

3.1.3. Learning knowledge of design constraints and design expectation

Both design constraints and design expectation play a crucial role in controlling the design process. A design expectation is a designer's prediction about a design attribute's value. Expectations may consist of a default value, a range of possible values, or a simple relation between design attribute values. Relationships between key design attributes and dependent design attributes are expressed as expectation rules by Chabot and Brown (1994). An expectation rule may be triggered when a value for a key attribute has been decided. These expectations and expectation rule(s) are an important source of design knowledge that can be learned from past design experiences and past design cases by induction (Chabot & Brown, 1994).

Design constraint knowledge provides the means for detecting design failures (Chabot & Brown, 1994). Although there are a large number of constraint types, categorizing them into hard or soft constraints aids in detecting design failures. Hard constraints are either satisfied or violated. If a hard constraint is violated, then the design decision becomes unacceptable. Values for soft constraints are bounded by the degree of error tolerance that is acceptable or allowable. Therefore, while expectation rules provide values to progress a design, constraint(s) test the values and detect any design failure.

Since different constraints and different expectation rules are called into play during a design process, Chabot and Brown use the mechanism of constraint inheritance as the machine learning method of knowledge compilation (Brown, 1991) whereby new knowledge of constraints is incrementally formed or updated.

3.1.4. Learning design decomposition knowledge

Liu and Brown (1992, 1994) posit that decomposition knowledge can be generated from design object knowledge, functional knowledge, design cases, design heuristics, general problem-solving knowledge, and domain knowledge. To extract knowledge that is appropriate for a given design context, Liu and Brown introduce the concept of "decomposition factors." A *decomposition factor* is a suggestion about how to partition an entity that may be a design object, a component, or a set of attributes.

The learning mechanism used is that of knowledge compilation (Brown, 1991). Knowledge compilation is a type of learning in which existing knowledge is converted into new forms with the intent to improve problem-solving efficiency.

The final decomposition knowledge is represented in a tree structure. Each node of the decomposition tree contains a problem description and a list of possible alternatives, competing decomposition hypotheses, as well as a set of links to subproblems, knowledge sources, interaction information among subproblems, and other relevant information.

From design object knowledge, Liu and Brown identified three types of interactions among the components of an object. The first type of interaction deals with explicit relationships among the components, while the second type of interaction arises from the relationships between the object's attributes and the components' attributes. The third type deals with interactions arising from the intercomponent relations giving rise to interactions among the subcomponents.

Representing the components and interactions as nodes and links, respectively, in a graph, several pivotal nodes with many links converging to them can be identified. Each pivotal node plus its connected nodes form a decomposition factor.

Design cases in which there are specific design problems, with their associated design solutions, can be another source for generating decomposition hypotheses. Given a design problem, a set of relevant design cases can be retrieved by using the object's case index. A suitable case is selected if every constraint of the current design problem matches the constraints of a design case to a predetermined degree; then the design case becomes a candidate for decomposition factor seed. All of the selected cases are converted into decomposition factors. If a selected case has a matching quality above some limit, it can be used for generating decomposition hypotheses.

3.1.5. Learning knowledge for performance evaluation

Performance evaluation of a design in the various stages of the design process often determines if the design should be progressed further. But performance evaluation is not an exact science. Multiple criteria that are conflicting in nature are used to evaluate the performance of a design or designs. Current methods of evaluation often are either too simple or too complex for configuration design decisions (Murdoch & Ball, 1996). Several researchers (McLaughlin & Gero, 1987; Murdoch & Ball, 1996) have posited that known configuration solutions in terms of their topological or geometric layouts, components, and materials can be analyzed and evaluated in terms of their performance so that good aspects of these designs may be reused and poor ones changed or discarded. That is, knowledge of performance evaluation can be learned from past design configurations. Several machine learning approaches (McLaughlin & Gero, 1987; Murdoch & Ball, 1996) have been developed to elicit the acquisition of design rules that map from the design solution space (i.e., structure or form) to the design behavior space.

McLaughlin and Gero (1987) formalize the relationship between decisions about values of design variables and their consequent performances as a mapping between a decision space and a performance space. Design variables can be represented as axes in a design (or decision) space, where each point in the space will represent a particular combination of design decisions regarding the design variables. By identifying the *best* feasible point in the performance space and mapping back to the decision space, the variables and their values that resulted in this performance can be identified. For performance space associated with multiple criteria, Pareto optimization is used to isolate the set of best solutions in terms of the criteria defining the performance space.

The knowledge acquired either through the Pareto/ID3 (McLaughlin & Gero, 1987) or GA/Kohonen Feature Map (Murdoch & Ball, 1996) is:

- Knowledge of the mapping between performance space (described by evaluation criteria) and the design decision space (described by design variables and their values).

3.1.6. Learning function knowledge and the causal models

Bhatta and Goel (1996) posit that generic teleological mechanisms (GTMs) are one type of design abstraction that can be learned from past designs through cross-domain analogies. Cross-domain analogy involves the recognition of similarity between two problems from two different domains and determining what knowledge to transfer and how to transfer between them. GTMs take as input the functions of a desired design and a known design and suggest patterned modifications to the structure of the known design that would result in the desired design.

3.1.7. Dynamic learning of implicit design knowledge

So far, the systems that use machine learning methods to acquire the design object knowledge for a given domain have been based on a fixed viewpoint that is predetermined by the knowledge engineer. Since it is not possible to completely predict designers' knowledge requirements, because each designer has different knowledge needs at different times and for different reasons, Kerr (1993) presents a new approach to utilize experiential knowledge called *customized viewpoint.*

The key concept behind this flexible approach to knowledge utilization is that it generalizes experiential knowledge directly from specific experiences, according to designers' knowledge needs, and subsequently utilizes this knowledge in design. To illustrate the concept, Kerr developed, tested, and evaluated the utility of the customized viewpoint approach within the realm of numerical design. PERSPECT is supported by four subsystems, namely, DESIGNER, ECOBWEB, S-PLUS, and GRAPHER (Duffy & Duffy, 1996; Kerr, 1993).

PERSPECT uses ECOBWEB, a concept formation system to generate and organize the generalization of a set of past designs (described by attributes and values) into a conceptual hierarchy. It classifies each example (past design) and incorporates it permanently into a hierarchy by incrementally changing the hierarchy's structure. The knowledge acquired by the approach is:

- Multiple forms of experiential knowledge of a domain (described by name, meaning, and units of attributes) in terms of empirical equations, numerical generalization, and heuristics. For example, empirical equations quantify the relationship between these attributes together with a measure of the unreliability of these equations.
- Multiple forms of implicit experiential knowledge through generalizations of past design information and identifying the most suitable generalization of past design that supports the *customized viewpoints* for a new design.

3.2. Design process knowledge

Learning about the design process provides just as an important wealth of knowledge as learning from past designs. A design process usually consists of design decisions made that results in design actions taken to progress the design. Design actions clearly interact with the evolving design. Learning about the design process invariably involves learning about the decisions made, the rationale for the decisions made, and the effect of that decision on the evolving design. Certain phases of the design process are exploratory in nature, and the use of control knowledge to manage the exploration is important.

Systems that capture design rationale (Gruber et al., 1991) and design history thus provide vital sources of knowledge for learning about the design process. While capturing design rationale usually involves recording successful design decisions, the design history records both successful and failed decision lines. Both successful decisions and failed decision lines are objects of learning about the process.

Knowledge of the design process can be captured in the form of design plans or as a hierarchy of activations (condition and action pairs) on the blackboard of a blackboard system (Erman et al., 1980; Hayes-Roth, 1985). In artificial intelligence (AI) parlance, a sequence of dependent actions is called a *plan.* As such, systems such as ADAM (Knapp & Parker, 1991), BOGART (Mostow, 1989), ARGO (Huhns & Acosta,1992), and DONTE (Tong, 1992) have used the AI approach of planning to describe and capture the design process, whereas only DDIS (Wang & Howard, 1994) has used the blackboard concept. Design plans usually do not capture failed lines of design actions, whereas the blackboard approach usually captures both the successful as well as the failed design actions, resulting in capturing the design history.

If the design plan is expanded to lower levels of abstraction, it also defines the hierarchical decomposition of the abstract actions into the primitive actions, the outcomes of which change the states of the evolving design. The evolving design is expressed initially in terms of functional requirements and at its lowest level of abstraction as design objects that collectively synthesize into a design configuration or structure. The design plan therefore captures not only the design history, design decisions (i.e., design intent and alternatives and rationale), and design strategies but also design object knowledge.

The knowledge change of design plans can be accomplished by the acquisition of new design plans, modifications of existing plans through design re-use, and generation of new plans or generalized plans from existing plans at the end of one or more similar design processes. The review of systems such as BOGART, ARGO, DONTE, CDA, and DDIS provides evidence of such knowledge change.

3.2.1. Acquiring design plans

A knowledge change in a design plan can be learned through the process of acquisition. This simply involves being told by the designer or recording the series of actions taken by the design, by inferring the design plan from a previous design case or solution (Duffy, 1997).

Both BOGART and ARGO record user inputs as a means of acquiring new plans. BOGART uses VEXED to record successive design steps in a tree-like design plan that consists of nodes representing design modules. Each module can be decomposed or refined into submodules by a catalog of "if-then" refinement rules provided by VEXED. If VEXED lacks any of these rules, it will learn the manual decomposition step by generalizing that step into a new rule. This learning facility within VEXED is provided by LEAP (Learning Apprentice).

ARGO acquires a plan as it solves a problem and is represented using an acyclic graph of dependencies among plan steps (instantiated rules). If one plan step adds an assertion that satisfies the condition part of a rule, the second step becomes dependent on the first. The dependency graph may contain independently solvable subproblems or dependent subproblems with justifications maintained by the truth maintenance system.

In ARGO, the design plans are represented as schemas of corresponding preconditions and postconditions that are represented as a database of assertions stored as slots of frames in a truth maintenance system. A module is represented as a collection of assertions describing its specification, components, interconnections, and so forth, each with a belief status of IN or OUT supported by a set of justifications. The OUT status is caused by actions that fail, and the corresponding rule instances are not included in the dependency graph for representing the design plan. Therefore, ARGO does not learn plans that incorporate failed lines of reasoning.

3.2.2. Acquiring case-dependent design plans

DDIS integrates both domain-based reasoning and case-based reasoning in its strategy for solving design problems. It uses case-dependent knowledge that it acquires from current design session(s) to supplement its domain-independent knowledge for future design(s) or redesign(s). DDIS therefore records all design actions as knowledge source activation records (KSARs) and the design history (a sequence of executed KSs and their bindings) on its design blackboard. By analyzing these records, DDIS abstracts case-dependent plans and goals. These plans and goals can be posted directly on the control blackboard by case-dependent control actions during subsequent design sessions. These case-dependent actions compete with case-independent knowledge sources at every design cycle to allow case-based reasoning to influence domain-based reasoning, so that past design actions leading to dead-ends or failures are avoided.

DDIS therefore generates case-dependent design knowledge after the completion of the design session. The knowledge generated consists of the following:

- control knowledge of a particular session is also abstracted to a global design plan and several redesign plans so that they can be used separately in a flexible manner according to new situations encountered;
- knowledge of constraint violations that can be applied to new cases in order to focus early on critical constraints that are most likely to cause problems.

3.2.3. Generalized design plans

BOGART, ARGO, and DDIS also generate generalized plans from several plans or by abstracting new plans. The details of how the generalized plans are generated through various knowledge transformers are given in Section 4.

3.3. Summary of the types of knowledge learned

Table 2 gives a summary of the various types of design knowledge that can be learned from the MLinD systems. The table shows the knowledge input I_k, the product/process knowledge learned O_k, and the reason for learning that knowledge, G_l.[1]

4. KNOWLEDGE TRANSFORMERS IN MLinD SYSTEMS

In this section, the purpose is to show the evidence of the knowledge transformers considered in Section 2 that MLinD systems use to transform input knowledge into new or modified design knowledge. Although these knowledge transformations are implemented in terms of various machine learning methods (either symbolic or subsymbolic), it is the nature of the characteristics of each type of knowledge transformation that makes their identification within the MLinD systems possible. Each machine learning method used may be either symbolic or subsymbolic in nature, depending on the nature of the representation of the input/output knowledge. In describing the various knowledge transformers, no distinction is made as to whether the knowledge transformed is product design knowledge or design process knowledge.

[1]This is usually inferred from the context of the design problem described.

Table 2. *Relationships between input knowledge, output knowledge, and learning goal*

Input knowledge I_k	Output knowledge O_k	Learning goal G_l
Instance(s) of past design(s) together with existing taxonomic or compositional knowledge	• Taxonomic knowledge of design concepts (e.g., BRIDGER) • Compositional knowledge of design concepts (e.g., NODES)	• Expedite synthesis of preliminary design concepts
Records of past designs in terms of attributes and attribute values	• Empirical knowledge of quantitative information (e.g., CONCEPTOR, NODES) • Design patterns of qualitative relationships (e.g., CONCEPTOR, NODES)	• Expedite preliminary definition of form and structure of design concept
Knowledge of current design constraints Records of failed constraints	• Knowledge of new/updated design constraints (e.g., Chabot & Brown, 1994) • Knowledge of anticipated crucial constraints (e.g., DDIS)	• Streamline design process by detecting and avoiding design failure • By checking crucial constraints early in the design, leading to shorter design cycle
Past design cases of design problems and corresponding solutions	• Design decomposition knowledge in terms of decomposition factors (e.g., Liu & Brown, 1992)	• Streamline design process by focusing on interrelated systems and/or components
Past design configurations and performance evaluation criteria	• Knowledge of mapping between performance evaluation space and design decision space (e.g., McLaughlin & Gero, 1987; Murdoch & Ball, 1996)	• Excellence driven to achieve better design
Past design concepts described by attributes and values	• Multiple forms of explicit design knowledge (e.g., PERSPECT) • Multiple forms of implicit design knowledge (e.g., PERSPECT)	• Excellence driven by utilizing knowledge from multiple sources
Records of design actions described by preconditions and postconditions	• Abstracted design plan (e.g., ARGO) • Case-dependent design plan (e.g., DDIS)	• Streamline design process by replaying a similar plan

4.1. Group rationalization (or clustering)/ decomposition (ungroup)

BRIDGER and CONCEPTOR are examples of machine learning systems that learn design concepts using the knowledge transformer of group rationalization or clustering. While CONCEPTOR uses the concept clustering system called COBWEB, BRIDGER uses ECOBWEB (Reich, 1993), which is an extension of COBWEB. Both machine learning systems for concept formation generate design concepts from the characteristics of similar past designs. Using past bridge designs as training examples, each system generates hierarchical classification structures that can be used to assist in the synthesis of similar bridge designs.

CONCEPTOR not only learns design concepts from past designs through the knowledge transformers of clustering (or concept aggregation—a term used in CONCEPTOR), it also derives numerical relationships (or concept characterization in CONCEPTOR's terminology) among the quantitative design attributes of a concept. Derivation as a knowledge transformer is described in Section 4.4.

COBWEB/ECOBWEB. To build the hierarchical structure, COBWEB/ECOBWEB integrates the processes of classifying examples and incorporating them into a hierarchy. It employs five operators to determine how best to incorporate an example into the hierarchy. Each resulting partition (classification) is evaluated using a utility function to determine the category utility (Gluck & Corter, 1985), which is a measure that quantifies the similarity between members of a partition. It selects the classification that results in the highest category utility value, incorporates the example permanently into the hierarchy, and generates the appropriate conceptual description that suits the new incorporated example. Each partition is described as a conjunction of attribute–value pairs, and each partition has a probability to indicate its frequency of occurrence in the training examples. As a result, a hierarchy represents only one concept, while the nodes in the hierarchy represent subsets of the concept.

Hence, in learning taxonomic-type design concept(s) from past designs, the knowledge change involved is the clustering of past designs into a hierarchical structure of concepts, each node in the hierarchy representing subsets of a main concept. Knowledge of the design artefact such as functionality, structure, or behavior is not explicitly represented. The knowledge of the artefact's decomposition structure or composite structure in terms of "part-of" links is not reflected in BRIDGER's/CONCEPTOR's classification hierarchy structure.

Another example where the knowledge transformer of clustering is used is in the context of learning design performance evaluation knowledge. The design solution space has to be categorized into different classes of solutions for the purpose of mapping them with the performance evaluation space. To elicit knowledge for performance evaluation, meaningful mappings between the decision space and the performance space modeled must be established. While McLaughlin and Gero (1987) concentrate on the solutions near the Pareto boundary as sources of knowledge in the evaluation space, Murdoch and Ball (1996) suggest that the entire solution set, from both the design space and the evaluation space, represents valuable design information that must be analyzed for effective reuse. The entire evaluation space and the design spaces must be analyzed to identify different classes of solutions and trends in design practice. Because of different areas of the design solution space and the performance space considered for mapping, McLaughlin and Gero (1987) chose ID3 to distinguish solutions that are Pareto-optimal (i.e., near the Pareto boundary) and those that are not, while Murdoch and Ball (1996) chose the clustering capability of the self-organizing neural network called the Kohonen Feature Map to categorize the entire design solution space.

ID3. The induction algorithm ID3 is used as a means of inferring general statements about the nature of solutions that exhibit Pareto optimal performance in terms of a set of performance criteria. The positive example set consists only of decision and performance data of solutions that are Pareto-optimal in terms of the chosen criteria. The negative example set is generated by combinations of design decisions that are inferior in performance. The heuristic rules that best represent the concept to be learned are those with the most positive examples and the least negative examples.

Kohonen Feature Map. The Kohonen Feature Map is a neural network that can learn clustering patterns unsupervised. That is, the mapping between criteria in the performance evaluation space and the configuration parameters (i.e., component and materials) in the design space can be clustered without precategorization of the design space.

Each solution consists of a set of parameterized components that can be applied as a training example to the network. The network uses an unsupervised learning algorithm to generate a mapping between the high-dimensional design space of component parameters and the neurons in the network. The map generated provides a topological (i.e., nearest neighbor) relationship among the component parameters. The network topology then can be inspected to identify clusters or archetypes that span the original set of design solutions. By decoding the nodes within each cluster back to the performance evaluation space, a mapping between the two spaces is achieved. The archetype solutions then are analyzed to identify the characteristics of configurations that contribute to high or low technical merit.[2]

4.2. Similarity/dissimilarity comparison

Knowledge change of a design plan through similarity comparison is made possible if there exists an original target design plan. The machine learning method used is learning by analogy, which involves a transfer of information/knowledge from a base domain/plan to a target domain/modified plan.

Having acquired the history of design decisions made in a previous design, BOGART (Mostow, 1989; Mostow et al., 1992) uses the derivational analogy method by Carbonell (1983, 1986) to change the design plan by reasoning from the previous plan. The derivational analogy method represents a problem-solving plan as a hierarchical goal structure, showing how and why each goal was decomposed into subgoals. It solves a new problem by replaying this plan top-down. When the subplan for a subgoal fails, the plan is modified by solving that subgoal from the user input of a new solution. By this process of similarity/dissimilarity comparison, a new design plan is constructed.

4.3. Association/disassociation

NODES learns/builds a model of the conceptual design linked by a compositional network of concepts through "part-of" and "kind-of" associations and a numerical network of characteristics through association between objects and formulas. In the compositional network the nodes denote objects or assemblies, and the arcs denote the directed relation (or association) "part-of" between two nodes. In the numerical network, nodes represent the characteristics of objects or formulas, and arcs represent the link or association between two nodes, one of which is a characteristic and the other of which is a formula in which the characteristic appears (Duffy & Duffy, 1996).

4.4. Derivation/randomization

The concept characterization phase of CONCEPTOR is an example of deriving new knowledge based on some dependency between them. After concept aggregation, CONCEPTOR derives two types of relationships within a concept: empirical formulas among quantitative design attributes and design patterns among qualitative attribute–value pairs that frequently appeared in past examples.

[2]Technical merit combines in one generic measure of design merit three fundamental criteria: performance (duty index), reliability (reliability index), and economy (cost).

For some design problems there is a need to apply the probability estimation function to acquire the probability estimate of each value of each unassigned design property (e.g., estimates of loading, the material behavioral properties are statistical in nature).

Ivezic and Garrett (1994) developed a system called NETSYN to learn the Bayesian *a posteriori* probabilities of design properties. NETSYN uses the feed-forward backpropagation neural network as the machine learning technique to acquire and represent the probability estimation function. The probability estimation function is acquired through inductive learning using past designs to train the neural network to estimate the desired probabilities. The trained network estimates Bayesian *a posteriori* probabilities.

To use conventional classifiers, one has to estimate the conditional probabilities $P(D|H_i)$ for each design property and the *a priori* class probabilities. The main difficulty lies in the elicitation of conditional probabilities $P(D|H_i)$ that reflect the actual design knowledge. This is estimated by assuming some idealized probability distribution (e.g., Gaussian distribution). The neural network approach estimates the Bayesian probabilities in a direct way, offering an approach where prior assumptions on probability distributions need not be made.

The computational model estimates the probability for each value of each property being used in a given design context. Each design context involves several design properties for which values have to be assigned. Therefore, the construction of NETSYN architecture is modular, that is, for each design property a neural network structure is assigned to act as a probability estimation function for that property.

4.5. Generalization/specialization

Knowledge derived through generalization has a greater problem-solving scope. This is because generalized rules or knowledge generally can be applied to a wider range of problems for a given domain or complex problem. Different types of knowledge related to the design product/process can be derived through the process of generalization. For example, NODES enriches its design knowledge base, called the *concept library*, by progressively accumulating solutions of problems defined within a particular domain. It uses generalization as a knowledge transformer so that new concepts are reflected in all of the concepts that are generalizations of that concept. By integrating with the DESIGNER system, numerical aspects of the concept (i.e., characteristics and associated formulas) can be analyzed. Both BOGART (Mostow, 1989) and ARGO (Huhns & Acosta, 1992) acquire generalized design plans from several plan instances or by abstracting new plans. Design actions interact with the evolving design. The interaction between design action and the design could be generalized into useful design rules.

Generalized design concepts. NODES generalizes knowledge from the most comprehensive concepts within a concept library to the less specific. Numerical parameter ranges and compositional knowledge are generalized to all of the associated superclasses to ensure that there is no contradiction between a particular concept and its specializations. The generalization mechanism that is responsible for the updating of knowledge is invoked automatically whenever a new concept is saved in the concept library.

When a design has been completed, the evolved model in NODES is used to increase its knowledge by acquiring the relevant knowledge of the new design. The mechanism involved is the decomposition of the design into its constituent concepts (or specializations), along with appropriate constituent and connective relations, and merging each concept with its corresponding library.

NODES uses a machine learning technique called *maximal conjunctive generalization* (MCG) (Dietterich & Michalski, 1983). MCG ensures that no item of knowledge is associated with a concept unless it is associated with all of the concepts that are a specialization of that concept. In terms of set theory, this means that the set of items of knowledge associated with a concept is the intersection of the sets associated with the specialization of that concept.

Generalized design plans. BOGART uses VEXED's ability (Steinberg, 1992) to interactively record decisions in terms of general rules that can be easily replayed in a new context, rather than specific operations that cannot be generalized. In BOGART, a design plan contains a node for each module. When the module is refined, the node is annotated with the name of the decomposition rule and the values of its parameters, and connected to a new child node for each submodule.

Generalized design rules. BOGART uses LEAP (Mitchell et al., 1990) to learn new design object knowledge. In the domain of circuit design, the training example consists of a description of the function to be implemented, a description of the known characteristics of the input signals, and a circuit provided by the user to implement the given function for the given input signals. LEAP generalizes the specific example into a new refinement rule. By using a variant of explanation-based learning (called *verification-based learning*), LEAP computes a refinement rule precondition (i.e., the left-hand side) by using its theory of circuits to analyze the single training example. LEAP explains (verifies) for itself that the circuit does in fact work. It generalizes from the example by retaining only those features of the signals that characterized this class of input signal. LEAP generalizes the right-hand side by verifying that the circuit correctly implements the desired function. The verification involves determining the general class of circuits and functional specifications to which the same verification steps will apply.

Specialized (compiled) design constraint knowledge. Design constraint knowledge is the primary method of detecting design failure (Chabot & Brown, 1994). Past design knowledge can be expressed as design expectation rules that relate the key design attribute with the dependent design attributes. When the key design attribute's value is decided, the design expectation rule is triggered. This results in the numerical or symbolic computation of the expected values of dependent design attributes. The expected values of either type (expected value range or expected symbolic value) are compared with the corresponding design attribute values. An *expectation violation* (Chabot & Brown, 1994) occurs when an inconsistency is noticed between the two sets of values, resulting in the creation of a DSPL[3] expectation violation structure. The information in the expectation violation structure is used by the Generic Object Knowledge Base (GOKB) Reasoner to transform the relevant constraining knowledge in the GOKB into a DSPL constraint. The knowledge compilation process consists of four sequential subprocesses of Reasoner, Transformer, Inheritor, and Executor. The Reasoner analyzes the role descriptors of the dependent (target) attribute into either potential numeric values or a member of a list of symbolic values and the structural descriptors of the design attributes. Constraints are inherited from the GOKB when a relevant explanation has been found by the Reasoner. The Transformer component supervises the transformation of the relevant GOKB knowledge into a DSPL constraint structure. The newly inherited constraint is tested by the Executor component. A "successful" test ensures that the new design attribute is valid and the newly inherited constraint knowledge is learned as a DSPL entity for future use.

Chabot and Brown therefore view constraint inheritance as a form of failure-driven learning that transforms a less efficient generalized deep object knowledge into surface knowledge that is highly specialized, tuned, and effective for the given design problem.

4.6. Abstraction/detailing

Abstraction in empirical equations. Abstraction in empirical equations may become necessary in the event that no useful empirical equations exist or because not enough attribute values are known. PERSPECT can be used to estimate the values of the unknown attributes. Using their own or PERSPECT's knowledge of design attribute dependency, designers using PERSPECT can define a perspective consisting of unknown attributes and related attributes, generate a viewpoint of experiential knowledge that can be used to find a past design or group of designs similar to the current design, and use associated similar attributes as values for the uninstantiated attributes in the current design.

If the domain model is too complex (i.e., described by too many empirical equations), designers can delete unwanted variables from empirical equations to generate simpler equations, which then can be used to estimate values of the design model. PERSPECT achieves this capability by the process of *abstraction.* Abstractions of empirical equations mean that dependent attributes can be assigned with fewer required attributes. DESIGNER can be used to determine the least influential input variable of the equation and suggest the variable as most suitable for deleting from the empirical equation.

Design plan abstraction. Both ARGO and DDIS use the knowledge transformation of abstraction to generate design plans. In ARGO, this task is accomplished by computing macrorules for increasingly abstract versions of the plan and inserting these rules into a partial order according to some abstraction relation. Macrorules, consisting of relevant preconditions and postconditions, are computed for each plan and stored in a partial order according to an abstraction scheme. These macrorules are built by compiling through the instances rules of the plan using a variant of explanation-based generalization (EBG) (Mitchell et al., 1986; DeJong & Mooney, 1986). The abstraction is accomplished by incrementally merging each set of edge macrorules into a set of cumulative macrorules for previously merged rules. The plan abstraction scheme consists of deleting all of its leaf rules that have no outgoing dependency edges, since these leaf rules are those that deal with design details.

At the end of each design session, DDIS abstracts the control knowledge recorded on the control blackboard to one global design plan and several redesign plans. The processes the DDIS uses to abstract design plans are as follows:

- All knowledge source activation records (KSARs) that modified the solution blackboard are identified and unnecessary design steps that led to unsuccessful alternatives or that did not contribute directly to the design process are filtered or removed.
- For each identified major action (KSAR), a case-dependent goal is created to prefer the same knowledge source or same type of action in the future.
- The major design actions are classified into design and redesign actions. Design actions are those that lead directly to the eventual solution. Redesign actions are those that are executed when a constraint violation is present on the blackboard. The goals corresponding to design actions are grouped into the design plan, while the redesign goals are grouped into redesign plans corresponding to each backtracking episode that resulted from constraint violations.
- The intentions of a global plan and case-dependent redesign plan are stated. The intention of a global plan is to generate design values for all of the design attributes and to satisfy all of the applicable constraints of the design. The intention of a case-dependent redesign plan

[3]Design Specialists and Plans Language (DSPL) is a domain-independent expert system building language for expressing routine design knowledge.

is to satisfy all of the unsatisfied constraints that triggered the redesign process.
- The critical constraints that were violated and caused backtracking are recorded so that they can be considered early in future designs.

Detailing to reconstruct design history. Unlike BOGART and ARGO, which rely on records of past design decisions in the form of a design plan, CDA first reconstructs from a similar solution a design plan using predefined rules (Britt & Glagowski, 1996). So while BOGART uses a derivational analogy to solve a new design problem, CDA uses a reconstructive derivational analogy (RDA) algorithm to automatically reconstruct design plan(s) from a large collection of past working design(s). Using the knowledge base of circuit design domain rules and information about the new circuit problem, CDA's reconstruction expert finds applicable rules, and selects and applies the preferred rule to the current circuit, adding design components each time until the final circuit meets the design requirement. In this process of detailing CDA reconstructs the design plan and acquires the rules for the composition of the design.

4.7. Explanation/discovery

DONTE illustrates the learning of control knowledge to explore the design space through the process of discovery about the design space. Through the process of discovery, IDEAL (Bhatta & Goel, 1994) demonstrates learning physical principles of a "concept" description from examples without knowing the target concept *a priori*. The process of discovery is generally considered to consist of two distinct phases: hypothesis formation and hypothesis testing.

Learning design control knowledge in DONTE. The discovery learning task is initiated by hypothesis formation in which a current hypothesis on the design space is represented as a set of subproblems that are presumed to be independent. Through design decisions made on these independent subproblems guided by control heuristics, interactions of subproblems are discovered and these interactions are aggregated into what is referred to as a macrodecision. The objective of a design decision is to minimize a cost evaluation function that favors certain design solutions (e.g., NAND gates are preferred over other gates). The current hypothesis is updated by the formation of the macrodecision, resulting in a new hypothesis about the design search space. Through this process of hypothesis formation of the design search space, information assimilation through each design decision made, and updating the hypothesis, DONTE learns control knowledge to optimally search the design space.

Learning models of physical principles in IDEAL. Using hypothesis formation on past designs' structure-behavior-function (SBF) models of physical devices, Bhatta and Goel (1994) show how behavior-function (BF) models of physical principles can be acquired for future use in design. Discovering physical principles from abstract design models of physical devices is implemented as a learning component of IDEAL (Integrated DEsign by Analogy and Learning).

The models of specific devices (SBF models) provide both the content and constraints for learning the models of physical principles (BF models) by incremental generalization over design experiences. In particular, Bhatta and Goel show that the function of a device determines what to generalize from its SBF model, the SBF model suggests how far to generalize, and the topology of functions indicates what method to use for generalization. By using content and constraints of the model, IDEAL is able to discover physical principles using fewer examples.

Table 3 gives a classification of MLinD systems in terms of the knowledge transformers used to generate new design knowledge and implementation of that knowledge change through the machine learning methods supplemented by other methods (e.g., Pareto optimization), which results in certain types of knowledge structures.

5. TYPES OF TRIGGERS FOR MACHINE LEARNING

The purpose of this section is to show evidence of what can trigger learning and when that trigger is likely to occur. Knowing what these triggers are and when these triggers initiate the learning process are important questions that must be answered if machine learning capability is to be incorporated into design support systems. To discuss these triggers by themselves and not relate them to the context of the knowledge learned and the knowledge transformer involved would not show the relationship between these elements of learning. Sections 5.1 to 5.3 give some examples of learning design knowledge under different types of triggers: namely, the retrospective, *in situ*, and provisional triggers that are implemented/implied in the MLinD systems reviewed. Since these examples do not represent exhaustively the range of the types of triggers and their related triggering events, Table 4 gives a summary of what can trigger learning and when these triggers occur in relation to the knowledge learned and the knowledge transformer involved in the MLinD systems reviewed.

5.1. Retrospective triggers

Retrospective triggers for learning design knowledge can occur at the end of a design process. The sources of knowledge for retrospective triggers are past designs and their corresponding past design processes. So, while learning about the design process may occur *in situ*, provisionally or retrospectively by analyzing the recorded design plan/design history, learning from past designs is triggered only retrospectively (i.e., at the end of the design process). Examples of product knowledge learned in retrospect were described in Section 3.1.

Table 3. *Knowledge transformers used in various MLinD systems and the related machine learning or other methods*

Knowledge transformer	MLinD systems involved	Machine learning/Other methods	Design knowledge represented
Group rationalization/ decomposition	• BRIDGER	• ECOBWEB/EPROTOS	• Hierarchical structure of concept/ subconcept
	• CONCEPTOR	• COBWEB	• Decision tree of rules for Pareto optimum design
	• McLaughlin & Gero (1987)	• ID3/Pareto	• Clusters or archetypes of design solution mapped to performance evaluation space
	• Murdoch & Ball (1996)	• Kohonen neural network/GA	
Similarity/dissimilarity comparison	• BOGART	• Derivational analogy	• Design plan as a hierarchical goal structure
	• DDIS	• Case-based reasoning	• Design plan/history
Association/disassociation	• NODES	• Semantic links in network	• Compositional network of concepts • Numerical network of characteristics
Derivation/randomization	• CONCEPTOR	• Concept aggregation	• Empirical formula among quantitative attributes. Design patterns among qualitative attributes
	• NETSYN	• Modular backpropagation neural network	• Bayesian *a posterior* probabilities of design properties represented as network of weights in neural network structure
Generalization/specialization	• NODES • BOGART/LEAP • DSPL	• Maximal conjunctive generalization (MCG) • Generalization using EBL generalization • Knowledge compilation through constraint inheritance	• Generalized rules of design concepts • Generalized design rules • Generalized design plan. Constraint rules
Abstraction/detailing	• PERSPECT • ARGO	• ECOBWEB/DESIGNER • Merging edge macrorules into cumulative macro by removing leaf rules	• Abstracted empirical equations • Abstract plan of macrorules
	• DDIS	• Identify, classify all activated KSARs into two types of design plans	• A global design plan and several redesign plans
	• CDA	• Reconstructive derivational analogy	• Detailed design plan built bottom-up
Explanation/discovery	• DONTE	• Hypothesis formation/hypothesis testing	• Discovery of macrodecision rule to reduce search
	• IDEAL	• Hypothesis formation/hypothesis testing	• Discovery of physical principles from abstract design models

5.2. *In situ* triggers

In situ triggers of learning occur during the design process, when design decisions are made. Design decisions are made in relation to the design object and/or design process. These design decisions may lead to a successful design action or to a failure. Learning can occur under such design decisions and actions. Some examples of *in situ* triggers implemented in MLinD systems are discussed below.

Failure in achieving behavioral specifications. Design adaptation is a common practice in conceptual functional design. Design adaptation usually occurs in several phases: in adapting a design retrieved from past design cases to satisfy the new behavioral specification, and in diagnosing and redesigning a failed design to achieve the desired behavior. Thus, Goel and Stroulia consider the design adaptation task as learning (Ashok & Stroulia, 1996). This process of learning takes place *in situ* as the design adaptation processes. Goel and Stroulia identify three types of diagnosis in design adaptation that could trigger the learning process during design:

- The design does not achieve the desired function of the device. The device fails to achieve the desired function because of incorrect specifications of one/more of the components.
- The design results in undesirable behavior. The undesirable behavior is due to the under/overspecification of the attribute of the component that influences its behavior.
- The specified structural component in the design has poor behavior. The component fails because of the overspecification of another component whose behavior has an adverse effect on the specified component.

Table 4. *Learning triggers in relation to knowledge learned and knowledge transformer involved*

Knowledge transformer, K_t	Knowledge learned, O_k	What triggers learning, T_{lw}	When is learning triggered, T_{lt}
Group rationalization/decomposition	• Taxonomic knowledge of design concepts	• New concept	• Retrospective
	• Clusters of design configuration map to performance evaluation space	• Performance trends in new design	• Retrospective
Similarity/dissimilarity comparison	• Knowledge of design plan	• New but similar design	• *In situ*
	• Knowledge of case-based design plan	• New design case	• *In situ*
Association/disassociation	• Compositional knowledge of design concepts	• New design configuration	• *In situ*
Derivation/randomization	• Empirical formula among quantitative design attributes	• New/updating empirical relationship(s)	• Retrospective
	• Design patterns among qualitative attribute–value pairs	• New/updating design patterns	• Retrospective
	• Posterior probabilities of design properties	• New knowledge of *a posterior* probabilities	• Retrospective
Generalization/specialization	• Generalized design concepts	• New concept saved	• *In situ*
	• Generalized design plans	• Module(s) in plan refined	• *In situ*/retrospective
	• Generalized design rules	• No existing design rules	• *In situ*
	• Specialized design constraint knowledge	• Constraint violation	• *In situ*
Abstraction/detailing	• Abstracted empirical equations	• Nonexistence of useful empirical equation or insufficient knowledge of attribute values	• *In situ*/provisional
	• Abstracted design plan by removing leaf nodes from plan	• New abstracted design process	• Retrospective
	• Abstracted from session control knowledge of the following:	• Past design cases to improve design process of similar design(s)	• Retrospective
	♦ global design plan		
	♦ related redesign plans	• Crucial constraints that triggered redesign process	• Provisional
	♦ constraint violations		
	• Detailed design plan reconstructed bottom-up	• No similar design plan existed	• Provisional
Explanation/discovery	• Search control knowledge	• Optimal design solution	• Provisional
	• Models of physical principles	• Functional-driven design	• Retrospective

They attributed the design failure in the three types of diagnosis to structural causes. Therefore, while knowledge of function to structure ($F \rightarrow S$) mapping is useful for new conceptual designs that may be learned retrospectively, knowledge of structure-behavior-function (SBF) models can be learned *in situ* during design adaptation. Each of the three different diagnosis tasks requires different schemes for accessing the internal behaviors that result in the device functions. In particular, task (1) requires the use of design functions as indices into the internal behaviors that result in the device functions; task (2) requires indices from the primary behaviors of the device; and task (3) requires the use of structural components of the device as indices into the internal behaviors in which they play a functional role. KRITIK2's SBF models (Ashok & Stroulia, 1996) support all three kinds of indexing schemes. This enables the system to flexibly access the internal behaviors relevant to the current diagnosis task and to thereby localize the diagnostic search.

Violation of design expectations. The DSPL system with constraint inheritance implemented by Chabot and Brown (1994) is an example of *in situ* triggers that occur when design expectations are violated. Whenever design expectation violations occur, constraint inheritance as a form of failure-driven learning is activated. When an inconsistency is detected in the evaluation of design expectation rules, the relevant design object knowledge is identified for knowledge compilation into new constraint surface knowledge through the constraint inheritance learning mechanism. The new constraint knowledge then is used to test the value of the design attribute for which there is an expectation. The addition of the new constraint knowledge to the existing design constraint knowledge base leads to expectation-failure–driven learning (Chabot & Brown, 1994).

Customized viewpoints. The concept of customized viewpoints is an example of learning design knowledge *in situ*.

Depending on the design perspective that best suits the designer's current problem-solving situation, PERSPECT can generate *in situ* multiple forms of implicit experiential knowledge through generalizations of past design information and identifying the most suitable generalization of a past design that supports the current *customized viewpoints* for the design.

5.3. Provisional triggers

Control knowledge. To explore the design space, DONTE learns by gathering and assimilating information and generalization during its use. The control knowledge to search the design space is learned provisionally in anticipation of reducing the complexity of the search. This is achieved by examining a small portion of the design space and generalizing and applying the information gained to control the search of the entire design space. DONTE achieves this by aggregating primitive fine-grained subproblems into larger macrodecisions when evidence gathered during the design exploration suggests that these subproblems interact.

Failed constraints anticipated in redesign. DDIS uses the blackboard framework to integrate case-based reasoning and domain-based reasoning. Of the knowledge modules/sources in the case-based reasoner, a module called *failure anticipator* checks for potential failures (i.e., violation of design constraints) and avoids them in the new design, either by recognizing paths leading to unsatisfactory results or posting information about constraints that were critical in a previous design. When appropriate, DDIS checks for design constraints during the design process. Whenever constraint violations are found, DDIS's failure anticipator marks them as the major constraints to be checked in a new design by placing them as goals on the blackboard. All other constraints are deactivated at this time. This action assures that the constraints most likely to be critical are checked as soon as they become checkable.

6. DISCUSSION

Prompted by Grecu and Brown's (1996) "Dimensions of Machine Learning in Design," this paper has intended to act as a stimulus for discussion and future work. Their paper raises issues regarding machine learning in design in an unstructured manner. These issues relate to "What might be learned?", "What are the methods of learning?", and "What can trigger learning?", amongst other issues. This paper introduced and provided evidence for a foundation of learning in design in terms of five elements: input and output of knowledge, knowledge transformers, goals/reasons for learning, and triggers in learning. Although it cannot be claimed that the number of MLinD systems reviewed was exhaustive, the foundation presented here shows that issues in machine learning in design can be studied in a systematic and structured manner that was not apparent in Grecu and Brown's "Dimensions of Machine Learning in Design."

For example, Table 2 shows clearly the relationship between the type of input knowledge required and the type of design knowledge learned and the reason for learning that knowledge, that is, given a particular type of input knowledge, what new design knowledge can be learned, and why that knowledge is learned. In this manner, Table 2 not only exemplifies the answers to the questions of what are/is the "elements supporting learning," "availability of knowledge for learning," and the "consequences of learning," but it shows that it is just as important to state what the dimensions of learning are and also to know and understand the relationships between Grecu's and Brown's dimensions.

Table 3 shows that the methods of learning in Grecu's and Brown's dimensions can be categorized into various knowledge transformers, as presented in this paper. These knowledge transformers represent the basic types of knowledge change and provide a basis of classifying the various machine learning methods implemented in MLinD systems.

Table 4 not only illustrates examples of what can trigger learning, but it also classifies these triggers into retrospective, *in situ*, and provisional triggers to provide answers to the question of "What can trigger learning?" and "When does learning occur?" These questions are not answered in isolation, but in the context of what is learned (i.e., the knowledge output) and how it is learned (i.e., the knowledge transformer involved).

Thus far, several answers to these basic questions, and supporting documentation for the foundation for machine learning in design (MLinD), have been derived from a review of published systems in MLinD. To complete the study, the foundation should be derived and further substantiated by analyzing knowledge change during generic design activities. This is part of the authors' ongoing work.

7. CONCLUSIONS

In summary, the paper has attempted to provide a foundation upon which to base the work of machine learning in design. Five key elements to the learning process have been presented: input knowledge (I_k), knowledge transformers (K_t), output knowledge (O_k), goals/reason for learning (G_l), and triggers of learning (T_l). A number of machine learning in design (MLinD) systems have been reviewed with a view to the above elements. From this it can be seen that there is considerable work being carried out in MLinD research and that the foundation presented in this paper provides a structure upon which to base, analyze, and build on that work. Thus, in response to Grecu and Brown (1996), this paper presents a foundation of learning as a logical and structured basis for the study of machine learning in design and a structure upon which to base fundamental questions (as raised by Grecu and Brown) to progress the field.

REFERENCES

Ashok, A.K., & Stroulia, E. (1996). Functional device models and model-based diagnosis. *Artif. Intell. Engrg. Design, Anal. Manuf. 10*, 355–370.

Bhatta, S.R., & Goel, A.K. (1994). Discovery of physical principles from design experiences. *Artif. Intell. Engrg. Design, Anal. Manuf. 8*, 113–123.

Bhatta, S.R., & Goel, A.K. (1996). From design experiences to generic mechanisms—Model-based learning in analogical design. *Artif. Intell. Engrg. Design, Anal. Manuf. 10*, 131–136.

Britt, B.D., & Glagowski, T. (1996). Reconstructive derivational analogy—A machine learning approach to automate redesign. *Artif. Intell. Engrg. Design, Anal. Manuf. 10*, 115–126.

Brown, D.C. (1991). Compilation—The hidden dimension of design systems. *Proc. 3rd IFIP WG 5.2 Workshop on Intelligent CAD, Osaka, Japan, Intelligent CAD III* (Yoshikawa, H., Arbab, F., and Tomiyama, T., Eds.) 99–108. North Holland, Amsterdam.

Carbonell, J.G. (1986). Derivational analogy—A theory of reconstructive problem solving and expertise acquisition. In *Machine Learning: An Artificial Intelligence Approach 2* (Michalski, R.S., Carbonell, J.G., & Mitchell, T.M., Eds.), 371–392. Morgan Kaufmann, Los Altos, CA.

Carbonell, J.G. (1983). Derivational analogy and its role in problem solving. *Proc. AAAI-83*, Washington, DC, 64–69.

Chabot, R., & Brown, D.C. (1994). Knowledge compilation using constraint inheritance. *Artif. Intell. Engrg. Design, Anal. Manuf. 8*, 125–142.

DeJong, G., & Mooney, R. (1986). Explanation based learning—An alternative view. *Machine Learning 1(2)*, 145–176.

Dietterich, T.G., & Michalski, R.S. (1983). A comparative review of selected methods for learning from examples. In *Machine Learning: An Artificial Intelligence Approach* (Michalski, R.S., Carbonell, J.G., & Mitchell, T.M., Eds.), 41–81. Morgan Kaufmann, Los Altos, CA.

Duffy, A.H.B. (1997). The "What" and "How" of learning in design. In *IEEE Expert, Intelligent Systems & Their Applications*, 71–76. IEEE Computer Society, New York.

Duffy, S.M., & Duffy, A.H.B. (1996). Sharing the learning activity using intelligent CAD. *Artif. Intell. Engrg. Design, Anal. Manuf. 10*, 83–100.

Duffy, A.H.B., & Kerr, S.M. (1993). Customised perspectives of past design from automated group rationalisations. *Artif. Intell. Engrg. 8(3)*, 183–200.

Duffy, A.H.B., Persidis, A., & MacCallum, K.J. (1995). NODES—A numerical and object based modelling system for conceptual engineering design. In *Knowledge Based Systems* (Edmonds, E.A., Ed.). Elsevier Science Publishing Ltd., Amsterdam.

Erman, L.E., Hayes-Roth, F., Lesser, V.R., & Reddy, D.R. (1980). The Hearsay-II speech-understanding system—Integrating knowledge to resolve uncertainty. *Comput. Surveys 12*, 213–253.

Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proc. Seventh Ann. Conf. Cognitive Science Society*, 288–287. Lawrence Erlbaum, Irvine, CA.

Grecu, D.L., & Brown, D.C. (1996). Dimensions of learning in agent-based design. In *Proc. Machine Learning in Design, Workshop* (*AID '96: Fourth Int. Conf. Artificial Intelligence in Design*) (Duffy, A.H.B., Maher, M.L., & Brown, D.C., Eds.)

Gruber, T., Baudin, C., Boose, J., & Weber, J. (1991). Design rationale capture as knowledge acquisition—Trade-offs in the design of interactive tools. *Proc. Eighth Int. Workshop on Machine Learning*, Evanston, IL.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artif. Intell. 26*, 251–321.

Huhns, M.N., & Acosta, R.D. (1992). An analogical reasoning system for solving design problems. In *Artificial Intelligence in Engineering Design*, Vol. 2 (Tong, C., Ed.), 105–143. Academic Press, New York.

Ivezic, N., & Garrett, J.H. (1994). A neural network-based machine learning approach for supporting synthesis. *Artif. Intell. Engrg. Design, Anal. Manuf. 8*, 143–161.

Kerr, S.M. (1993). *Customised viewpoint support for utilising experiential knowledge in design.* PhD Thesis. University of Strathclyde.

Knapp, D., & Parker, A. (1991). The ADAM design planning engine. *IEEE Trans. Comput.-Aided Integrated Circuits and Syst. 10(7)*.

Leith, P. (1990). *Formalism in AI and Computer Science.* Ellis Horwood Limited.

Li, H. (1994). *Machine Learning in Design Concepts*, Computational Mechanics Publication.

Liu, J., & Brown, D.C. (1992). The generation of decomposition knowledge for near routine design. In *Applications of Artificial Intelligence in Engineering VII* (Grierson, D.E., Rzevski, G., & Adey, R.A., Eds.). Computational Mechanics Publication.

Liu, J., & Brown, D.C. (1994). Generating design decomposition knowledge for parametric design problems. In *Artificial Intelligence in Design '94* (Gero, J.S., & Sudweeks, F., Eds.), 661–678. Kluwer Academic Publishers.

McLaughlin, S., & Gero, J.S. (1987). Acquiring expert knowledge from characterised designs. *Artif. Intell. Engrg. Design, Anal. Manuf. 1*, 73–87.

Michalski, R.S. (1993). Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning 11*, 3–43.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based generalisation—A unifying view. *Machine Learning 1(1).*

Mitchell, T., Mabadevan, S., & Steinberg, L.I. (1990). LEAP—A learning apprentice for VSLI design. In *Machine Learning, An Artificial Intelligence Approach, Vol. III* (Kodratoff, Y., & Michalski, R., Eds.), 271–301. Morgan Kaufmann, Los Altos, CA.

Mostow, J. (1989). Design by derivational analogy—Issues in the automated replay of design plans. *Artif. Intell. 40*, 119–184.

Mostow, J., Barley, M., & Weinrich, T. (1992). Automated reuse of design plans in BOGART. In *Artificial Intelligence in Engineering Design*, Vol. 2 (Tong, C., Ed.), 287–332. Academic Press, New York.

Murdoch, T., & Ball, N. (1996). Machine learning in configuration design. *Artif. Intell. Engrg. Design, Anal. Manuf. 10*, 101–113.

Persidis, A., & Duffy, A. (1991). Learning in engineering design. In *Intelligent CAD III* (Yoshikawa, H., Arbab, F., & Tomiyama, T., Eds.), 251–272. Elsevier Science Publishers.

Reich, Y. (1993). The development of BRIDGER—A methodological study of research in the use of machine learning in design. *Artif. Intell. Engrg. 8(3)*, 165–181.

Steinberg, L.I. (1992). Design as top-down refinement plus constraint propagation. In *Artificial Intelligence in Engineering Design*, Vol. 1, (Tong, C., Ed.), 251–272. Academic Press, New York.

Tong, C. (1992). Using exploratory design to cope with design problem complexity. In *Artificial Intelligence in Engineering Design*, Vol. 2 (Tong, C., Ed.), 287–332. Academic Press, New York.

Vancza, J. (1991). Improving design knowledge by learning. In *Intelligent CAD III* (Yoshikawa, H., Arbab, F., & Tomiyama, T., Eds.) 289–305. Elsevier Science Publishers.

Wang, J., & Howard, H.C. (1994). Recording and reuse of design strategies in an integrated case-based design system. *Artif. Intell. Engrg. Design, Anal. Manuf. 8*, 219–238.

Siang-Kok Sim is currently a Senior Lecturer at the School of Production and Mechanical Engineering, Nanyang Technological University. His main research interests are knowledge-based conceptual design, machine learning in design, neural networks, and multiagent systems. He is currently pursuing a part-time Ph.D. programme at the University of Strathclyde, Glasgow, Scotland.

Alex H.B. Duffy is presently a Senior Lecturer and Director of the CAD Centre, University of Strathclyde, Glasgow, Scotland. His main research interests are knowledge-based conceptual design, product and design knowledge modeling, machine learning and past design utilization, performance measurement and design productivity, and design coordination. He is the leader of a European-wide network working in design coordination. He is on the advisory board of *AI in Design*, *Engineering Design*, and Concurrent Engineering conferences and on the editorial board of *Research in Engineering Design and Design Computing.* He organized and chaired the Engineering Design Debate on design productivity (EDD '96) and is a member of the IFIP Technical Committee Working Group 5.8 in product structuring and documentation.