ELSEVIER

Artificial Intelligence in Engineering 15 (2001) 153–164

ARTIFICIAL INTELLIGENCE in ENGINEERING

www.elsevier.com/locate/aieng

Knowledge modeling for design decisions

F. Mili[a,*], W. Shen[a], I. Martinez[b], Ph. Noel[b], M. Ram[b], E. Zouras[b]

[a]*School of Engineering and Computer Science, Oakland University, Rochester, MI 48309-4478, USA*
[b]*DaimlerChrysler Corporation, Auburn Hills, MI 48326-2757, USA*

Abstract

In this paper, we share our experience in modeling and representing design knowledge relevant for engineering design decisions. We define an object model where classes are used to capture design standards and requirements relevant to designed objects. The traditional object model is customized to the representation of design knowledge in two major ways: (1) Classes representing design objects are augmented with design validation information. (2) Associations between classes are made explicit and used to reduce the redundancy and maintain the consistency of the knowledge. We define the semantics of the resulting object model and formulate the axioms that define its consistency. The object model is defined in the context of stamping design knowledge.

Keywords: Engineering design; Knowledge modeling; Class constraints; Model integrity

1. Introduction

The activity of engineering design can be defined to encompass a variety of activities [1]. We use it to refer to all activities aiming at generating and refining detailed product descriptions prior to their physical realization [2]. The process of engineering design is an analytic step used to improve the quality of the final product and to reduce the time and resources needed for the final production [3]. The improved quality results from the potentially large number of candidate solutions considered during design. Descriptions of these solutions are generated, refined, analyzed, validated, and compared prior to any physical production of a selected candidate. The reflective activities of analysis and evaluation during design allow the early discovery and correction of problems, thus reducing the occurrence of trial and error, and saving time and cost during the production stage.

Because design is in essence a modeling activity, its effectiveness and efficiency are a direct function of the modeling language used [4]. The level of abstraction and the expressive power of the modeling language used to describe designs dictate the ease with which multiple alternatives can be generated and the amount and accuracy of the insight that can be obtained from them. Engineering design can greatly benefit from computer support [5–11]. Traditional design support systems consisted of geometry-based CAD systems. The amount and quality of support provided by geometry-based CAD systems is limited for a number of reasons. The main reasons lie in the narrowly focussed expressive power of geometry and in the wide semantic gap between the geometry and the abstract features of a product. There is a consensus in the engineering design community on the need for more comprehensive design support systems [12–18] and for a better integration of the reflective activities of design validation and evaluation with the active steps of design generation and refinement [19–24]. The project we describe here shares this general goal. It aims in particular at supporting design refinement by providing designers with all the information necessary to make decisions and to validate and justify these decisions. We discuss issues related to the modeling and the management of the knowledge used to support design activities. In particular, we present an object model used to capture the knowledge of interest, define the semantics of the language, and introduce axioms expressing modeling invariants.

This paper is organized as follows. In Section 2, we briefly introduce the problem and explain the general approach used. In Section 3, we provide a detailed presentation of the knowledge model with its constructs, its syntax and semantics. We also formulate a set of axioms capturing the integrity and consistency of the model. In Section 4, we summarize and discuss future extensions of this work.

2. Problem, approach

2.1. Problem

Two major issues need to be addressed to enable a

* Corresponding author.
E-mail address: mili@oakland.edu (F. Mili).

0954-1810/01/$ - see front matter
PII: S0954-1810(01)00013-9

comprehensive support of engineering design: (1) Raising the level of abstraction of CAD systems from low-level geometry towards high level concepts drawn from the design domain; (2) Integrating the evaluation and validation activities with the design activities. The multi-disciplinary nature of engineering artifacts and the amount of expertise involved make it impossible for the designer to thoroughly evaluate a design by simple inspection without outside support. As CAD systems move towards feature-based and object-oriented representations, providing an integrated decision support becomes within reach. The integration of validation and evaluation support with generation and modification support requires the existence of repositories of validation requirements and evaluation criteria. Engineering domains have the advantage of being highly documented with relatively formalized information. In addition, the recent trend of knowledge management has led a number of corporations to capitalize on corporate knowledge by eliciting it and recording it [25,26]. This creates additional sources that are proprietary and more focussed. In this paper, we use our experience with one such effort to motivate and illustrate our approach. We present the case study and then base all discussions and examples on it. In the concluding section, we discuss, among other things, the scope of the approach and its applicability to other domains and other applications.

2.2. Case study

A knowledge management effort was initiated a few years ago at DaimlerChrysler (then Chrysler). Selected units were given the mission to author their *books of knowledge*. The resulting books are intended to be live, up to date documents synthesizing the corporate expertise in the form of standards, procedures, and best practices. This expertise, once recorded, becomes more easily reviewed, shared, and used. The knowledge collected is, for the major part, knowledge relevant to the main business of the company: design and manufacture of automotive systems.

In addition to being valuable in their own right, the books represent an important stepping-stone towards an integrated system support of evaluation and validation activities during design. The stamping book of knowledge, for example, includes a major section defining constraints and criteria on the dimensions and shapes of all the parts that are manufactured by a stamping process. Considerations of feasibility and cost of the stamping process and considerations of quality of the stamped product drive these constraints and criteria. The representation and organization of the knowledge in the books has been initially crafted so as to facilitate its retrieval and readability by the authors, the reviewers, and other human readers from the same unit. This knowledge needs to be re-examined with the prospect of using it by a design support system. We discuss below issues and criteria directly relevant to the selection of a modeling language for this knowledge.

2.3. Scope and approach

The long-term goal of this project is to make use of the knowledge collected in the various books by enforcing its standards and recommendations during the design process. The books authored by the different units cover various aspects of corporate knowledge and corporate operation. Some of this knowledge is directly relevant to design. There was a deliberate choice not to inhibit the book contributors by letting them express the knowledge in whatever form they are comfortable with rather than imposing on them an additional burden of form. This separation between contents and form allowed also a staged development of the books and delayed the decision about form until sufficient information is available to make it.

With the first goal of eliciting the knowledge accomplished, we set to rethink the form of the knowledge in light of its contents and its planned and potential uses. A major underlying concern has been to preserve as much as possible the distributed ownership of the knowledge and the ease with which it is read, understood, and updated, while also making the knowledge accessible and usable by an automatic decision support system. In particular, we have used the following criteria:

- The books must remain separate, independent, and autonomous. This condition is necessary to ensure that the knowledge in the books truly reflects the latest innovations and best practices in each domain.
- The books must remain descriptive. We have found that the knowledge elicited in the books is descriptive in nature. It states the conditions that ideally should hold without prescribing fixes for those cases where they do not. It is important to let the domain experts state what they know best. Fixes for violated constraints depend on the processes, the context, and many other considerations outside their jurisdiction.
- Changes in the books must be reflected within the design support system with little or no delay.

The general solution adopted is illustrated in Fig. 1.

In this figure, each source represents the unit authoring and maintaining its book. This reflects the full ownership of the books by their corresponding units. Because the design support system needs timely access to the knowledge originated from the different books, it is networked to the books via a *knowledge extraction and transformation component*. This component extracts relevant knowledge from the different sources, and integrates it together into the design support system's knowledge base. In the process, the knowledge may be transformed into efficient prescriptive processes. The use of a level of indirection in the form of the knowledge extraction and transformation component introduces flexibility and relative independence, yet still requires that the knowledge in the books have some degree of formality. We discuss now the knowledge model

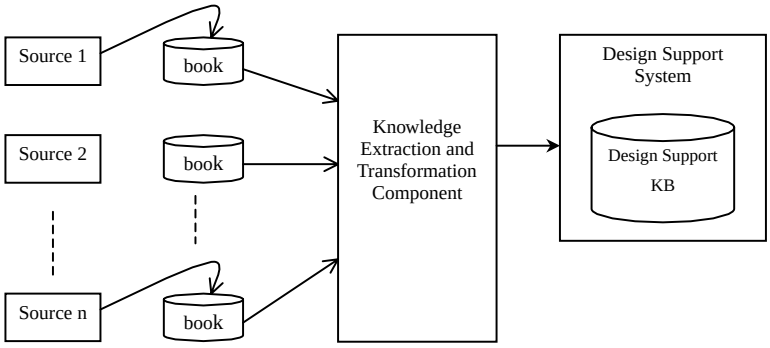


Fig. 1. System architecture.

developed for the revised versions of the books. Because we have so far worked with the Stamping Book of Knowledge, we have tailored the model to the stamping knowledge. The model will later be refined and adapted as more books are integrated.

Fig. 2 shows a sample 'page' of the stamping book. It is basically a set of constraints attached to the given panel.

3. Knowledge model: syntax, semantics, and integrity

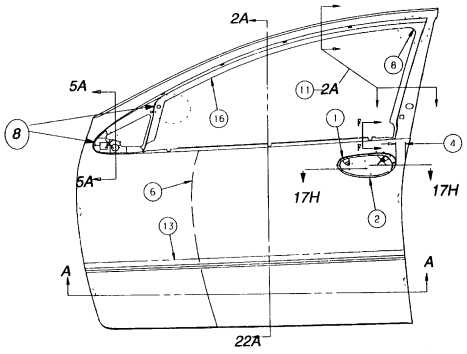
As illustrated by the sample data from the case study, the knowledge of interest is a set of feasibility and quality motivated requirements associated with objects of design and related concepts. In stamping, the objects of design are the panels that are manufactured by a stamping process. The panels are generally associated with and identified by the part that they belong to. For example, there are two major panels that compose a door: an outer door panel and an inner door panel. The knowledge model is organized around these major concepts of parts and panels represented by classes, and the concepts of design requirements associated with these classes. We define the knowledge model progressively by defining the following building blocks:

- The class construct.
- The class–subclass association.
- The class-feature association.
- The assembly-part association.

For each construct, we motivate the need for it, define its syntax and its semantics. At the same time, we formulate a set of axioms necessary for the integrity of the model.

3.1. The class construct

We use the classes primarily to represent the objects of design, i.e. parts, panels, and related concepts. Traditionally, classes have been used to encapsulate the state (attributes) and behavior (methods) of the classes of interest. In the



Rule No.	Text	Formula	Rationale
1.	The Side view Radii in the corners of the door handle depression should be greater than the depth of the handle opening.	Radii >= 22 mm	Causes mouse ears – low spots.
4.	The distance from door handle depression to the rear cutline should be maximized.	Distance >= 50mm	Causes mouse ears – low spots.
6.	There should be curvature in a top to bottom section through the body of the door.	Top to bottom Curvature >= # 2 Sweep.	Surface stretch.

Fig. 2. Sample of information in book of knowledge.

context of a design system, the attributes of an outer door panel would include all of those attributes that define its geometry, material, history, and so forth. Methods can be used to define derived properties. The distinction between attributes and methods is made solely on the basis of the level of granularity of the underlying representation. We choose, therefore, to use the term *property* to encompass both of them as is done in other sources [27]. For example, the information of interest on an outer door panel includes such things as:

- The angles at the corners of the panels must not be sharper than 50°.
- The length of the panel should not exceed 500 mm.
- The door handle depression must be away from the rear cut-line by at least 25 mm.

It is this and similar information that constitutes the design knowledge associated with the parts and panels. We expand the concept of classes by adding a 'compartment' (in UML terminology [28]), that we name *validation constraints*. The class construct we are interested in captures the constraints of objects of the class. The validation constraints are expressed in terms of properties of the class objects (angle, length, curvature, etc.). These properties used directly or indirectly in the expression of the constraints are recorded as part of the class definition.

3.1.1. Syntax

This class concept is illustrated in the example shown in Fig. 3. A few things are of notice about this class. First, the set of properties listed is by no means a comprehensive representation of the state of the class. We only list those properties that are relevant to the expression of the validation constraints. Second, the design constraints are not fully described here. They are only referenced. The constraints are represented as instances of their own class: constraint.

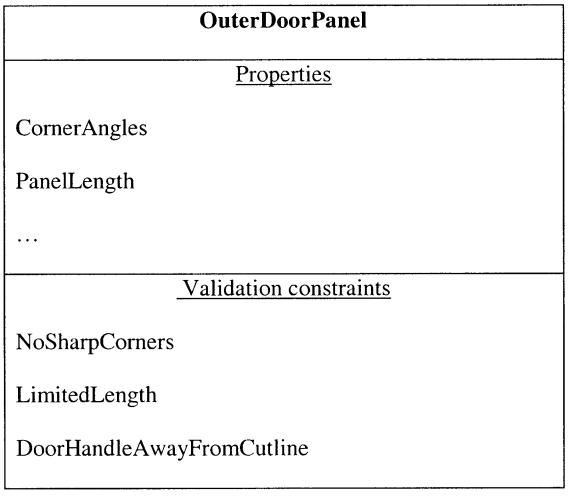


Fig. 3. Sample class.

We illustrate the class constraint by showing the softCorners instance in Fig. 4.

The properties used to describe constraints reflect the intended uses of these constraints. They are meant to be directly authored and illustrated by human experts; examined and understood by human designers; checked for design compliance by computer-based design support systems; and retrieved and managed by a knowledge management system. As such, they are represented and documented in a way that meets all of these needs. The defining properties consist of a name, a context, and a formula. The *name* is simply an identifier used to refer to the constraint. Each constraint is associated with one unique *context*. In the above example, the constraint applies to all panels, and is therefore associated with the 'general' context Panel. The property *formula* captures the formal expression of the constraint in a way that allows its automatic verification. In the examples, we use a first order logic (FOL) like syntax. Depending on the expressive power needed and other considerations, similar languages such as Object Constraint Language [29] could be used. Regardless of the language used, the atomic formulae must be drawn from properties defined in the context. The set cornerAngles( ) for example must be known (defined or inherited) in the context of a panel.

The properties used for 'human user' consumption are self-explanatory. In a graphic design setting, graphics and illustrations play a critical role both in expressing and in conveying the meaning of a constraint.

The management properties are used to maintain a documentation of the reason why the constraint is mandated, the premises underlying it, and other bookkeeping information. The *rationale* of a constraint is often expressed in terms of properties warranted by complying as well as faults resulting from violation. The *premise* further refines the information by stating the (implicit) assumptions underlying the constraint, i.e. the conditions under which the constraint does deliver the results expected and prevents the faults associated with its violation. Should a change in technology or practice conflict with the premises, the constraint needs to be reviewed and possibly revised. The property *reference* provides pointers to relevant technical references. In the current example, such references may include documents on the properties of the alloy X used as well as documents on the relationship between material properties and stamping processes for example. In the above example, if the alloy used for the panels is changed, this constraint may no longer be accurate. It needs to be reviewed and revised or dropped. Overall, the management properties are useful in identifying when constraints may need to be reviewed, and in identifying the type and level of revision required.

3.1.2. Semantics

Before proceeding further, we briefly discuss the meaning of the class construct. A class is defined by its properties and its constraints. The set of properties P_i of type T_i (P_i: T_i)

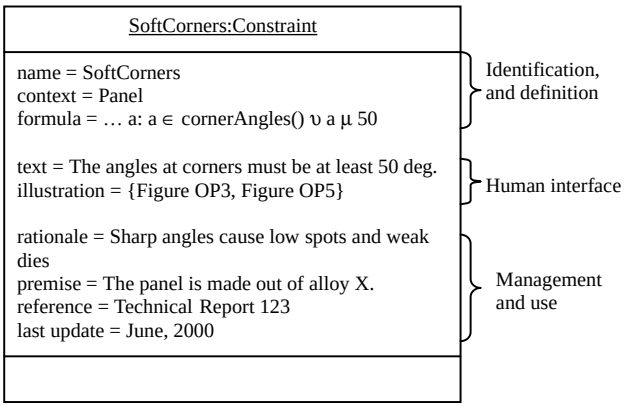


Fig. 4. Sample design constraint.

capture the working design space which we call W and define as $T_1 \times T_2 \times \ldots \times T_n$. Because the only properties included in the classes are those referenced in the constraints, the working space is only composed by those properties whose value affects compliance with norms and standards. The space W defines the set of all possible decisions, whether compliant with the constraints or not. When combined with the set of validation constraints, this information captures the target space T_g, that we define as $T_g = \{s \mid s \in W \text{ and } (C_1 \wedge C_2 \wedge \ldots \wedge C_k)(s)\}$. Note that the expression $(C_1 \wedge C_2 \wedge \ldots \wedge C_k)(s)$ is an abuse of notation. It is used as a short hand for $(C_1 \cdot \text{formula(s)} \wedge C_2 \cdot \text{formula(s)} \wedge \ldots \wedge C_k \cdot \text{formula(s)})(s)$.

There are other approaches in the object-oriented literature in which class constraints are used. Typically, these approaches interpret the constraints as class invariants, met by all instances at all times. In our design context, we need a more flexible configuration. Because design is a progressive refinement process that is non-monotonic. Designs in progress are generally non-fully compliant. This raises the need to accept all designs (W), yet identify those that are compliant (T_g). Furthermore, because design is a progressive refinement process, designs in progress are generally not fully defined (not all variables have been set); as a result, they represent sets of elements of the space rather than individual elements. These two concepts are summarized below and discussed in more detail in Ref. [17]. We formalize the definition we have given for the classes and formulate associated axioms.

Definition 3.1.1. Definition of class semantics. Given a design class D defined by $D = \langle \mathsf{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathsf{C} = \{C_j\}_{j=1\ldots k}\rangle$ where P is a set of properties and C a set of constraints, the class defines a pair of sets $\langle W, T_g\rangle$ as follows:

$$W = T_1 \times T_2 \times \ldots \times T_n$$
$$T_g = \{s \mid s \in W \text{ and } (C_1 \wedge C_2 \wedge \ldots \wedge C_k)(s)\}$$

The instances of the design class D subsets of W are called instances of D. The subsets of W that overlap with T_g are called valid designs. The subsets of D are correct designs.

The sets defined above are illustrated in Fig. 5.

For the definition above to be operational, the constraints C_j must have well-formed formulae in the space W. This condition is captured by the following *closure axiom.*

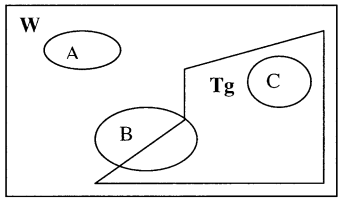
Closure axiom 3.1.1. Given a design class D defined by $\langle \mathsf{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathsf{C} = \{C_j\}_{j=1\ldots k}\rangle$, the formula of each of the constraints C_j is a well-formed formula whose atomic formulae are drawn from the set P.

Further, we require that classes have feasible target spaces. In other words, the constraints associated with a class must not be conflicting with each other. This is captured by the feasibility Axiom.

Feasibility axiom 3.1.2. Given a design class D defined by $\langle \mathsf{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathsf{C} = \{C_j\}_{j=1\ldots k}\rangle$, the target space T_g must not be empty.

3.2. Subclasses and inheritance

One of the flagships of object orientation is its concept of subclass and its associated properties' inheritance. A class' properties are automatically inherited by its subclasses; they can also be refined within the subclasses. We define our use of specialization adapting it to the definition of classes and



W: Working space **Tg**: Target Space
A: design **B**: valid design **C**: correct design

Fig. 5. Sample design constraint.

to the use of constraints. In Section 3.2.1, we define the syntax for declaring a specialization relationship and for adding and refining properties and constraints. In Section 3.2.2, we define the semantics of specialization and elicit the axioms restricting properties and constraints refinement. We limit our treatment here to simple inheritance, although the results can be easily combined with other approaches specifically addressing multiple inheritance [30,31].

3.2.1. Syntax

We illustrate the syntax of specialization through an example, illustrated in Fig. 6. We consider the class Panel defined below. The class Panel is defined as a subclass of the root class DesignClass. As such, all properties and validation constraints associated with DesignClass are also applicable to Panel. Further, the class OuterPanel is defined as a specialization of the class Panel. It inherits all the properties and constraints defined under Panel or its ancestors. In addition, OuterPanel defines its own properties and constraints.

Note that the two properties defined in the subclass OuterPanel relate to the two constraints added. The constraints remain the driving factor in the definition of classes. Properties and constraints inherited from superclasses can also be refined within the specialized class. The refinement of properties is simply done by redefining a property with the same name. We illustrate in Fig. 7 the refinement of a constraint using the hypothetical example of a Rounded Panel where the requirement on the corners' angle is more stringent than the softCorners constraint used for the rest of the panels. We turn now to the semantics of specialized classes and the rules for defining them.

3.2.2. Semantics

We have captured the meaning of classes in terms of a pair of spaces $\langle W, T_g \rangle$, where W is the working space, and T_g is the target space. Because this definition is state-based, the concept of sub-classing must also be state-based, and thus must be consistent with sub-setting. In other words, given a class D defining the pair of spaces $\langle W, T_g \rangle$, and given a class $D' : D$ defining a pair of spaces $\langle W', T_g{}' \rangle$, we must have pair-wise inclusion between the spaces, i.e. $W' \subseteq W$ and $T_g{}' \subseteq T_g$. This relationship holds strictly when no new properties have been defined within the specialized class. When D' defines new properties, the inclusion relationship is between the projections of the spaces of the specialized class and the spaces of the generalized class. We state this as an axiom and define the rules of specialization in a way consistent with this axiom.

Specialization axiom 3.2.1. Given a design class D defining the pair of spaces $\langle W, T_g \rangle$, and given a design class $D' : D$ defining $\langle W', T_g{}' \rangle$, the projection of W' over W is a subset of W, and the projection of $T_g{}'$ over W is a subset of T_g.

In order to define the rules of specialization, we need to define the working space and the target space of a class defined by specialization. We consider a class D' specialization of D, where properties l through n are redefined (refined) and properties $n+1$ through n' have been added. In terms of constraints, we only consider the addition of new ones for now.

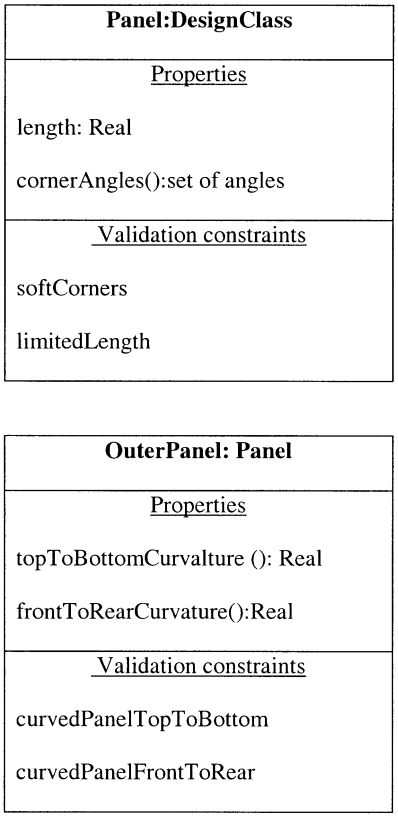


Fig. 6. Specialization.

Definition 3.2.1. Semantics of specialized classes. Given a design class D defined by $D = \langle P = \{P_i : T_i\}_{i=1\ldots n}, C = \{C_j\}_{j=1\ldots k} \rangle$, and given a design class defined by $D' =: D, \langle Q = \{P_l : U_l, \ldots, P_n : U_n, Q_{n+1} : U_{n+1} \ldots Q_{n'} : U_{n'}\}, E = \{C_j\}_{j=k+1\ldots k'} \rangle$, the class D' defines the following pair of spaces $\langle W', T_g{}' \rangle$:

$$W' = T_1 \times T_2 \times \ldots \times T_{l-1} \times U_l \times \ldots \times U_n \times U_{n+1} \times \ldots \times U_{n'}$$

$$T_g = \{s \mid s \in W \text{ and } (C_1 \wedge C_2 \wedge \ldots \wedge C_{k'})(s)\}.$$

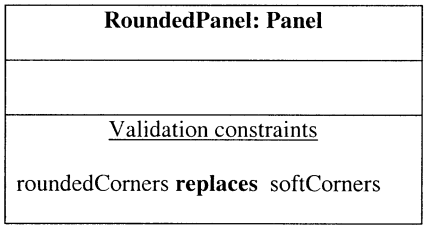


Fig. 7. Constraint specialization.

We state now the axioms regulating the specialization, first for the properties, and then for the constraints. We use the usual sub-typing relationship for tuples (e.g. see Refs. [32,33]) to regulate the validity of property addition and redefinition.

Property specialization and addition axiom 3.2.2. Given a design class D defined by $D = \langle P = \{P_i : T_i\}_{i=1\ldots n},\ C = \{C_j\}_{j=1\ldots k}\rangle$, and given a design class defined by $D' := D$, $\langle Q = \{P_l : U_l \ldots P_n : U_n,\ Q_{n+1} : U_{n+1} \ldots Q_{n'} : U_{n'}\},\ E = \{C_j\}_{j=k+1\ldots k'}\rangle$. For each i in $l \ldots n$, U_i must be a subtype of T_i.

The refinement of inherited constraints and the addition of new constraints are very similar. In order to be compliant with the specialization axiom, the set of constraints of the specialized class must imply the set of constraints of the generalized class ($T_g{}' \subseteq T_g$). When constraints are added, the implication is automatic. Care must be taken when constraints are refined (replaced). Because replacement is done on a one on one basis, it is sufficient to ensure that the replacing constraint is more stringent than the one being replaced.

Constraint specialization axiom 3.2.3. Given a design class D defining the pair of spaces $\langle W, T_g\rangle$, and given a design class $D' : D$ in which we have a clause C' *replaces* C, C'·formula $\Rightarrow$ C·formula.

The refinement of constraints is often driven by more stringent requirements. The RoundedPanel example is an illustration of this situation. This is not the only possible scenario, though. In practice, a constraint may need to be redefined for other reasons than its logical formula. Any of the attributes of the Validation Constraint can be redefined (e.g. rationale, text, illustration, reference, etc.). It is conceivable to create a dependency between the attributes to impose pre-defined sequence of changes. For example, the value of the attribute Last Updated can be changed automatically; the rationale may need to be updated to reflect a change in the formula; some changes in a formula may require changes in the illustration, etc.

So far, we have focussed on the local aspects of specialization. Globally, specialization creates a lattice (here a tree) structure. The non-cyclic and rootedness aspects of this structure are essential properties.

Specialization tree axiom 3.2.4. The specialization association defines a rooted tree structure. Design Class is the root of the tree.

3.3. Part-feature association

This generalization/specialization association proved to be a very powerful tool for reuse [34]. In general, associations are useful to the extent that they provide us with an economy of expression and an added power of inference. The power of expression is obtained through the modular definition of classes. A class can be defined as a special case of more general classes. The properties of a class do not need to be repeated for its subclasses.

The specialization association goes a long way towards adding modularity to the design knowledge. The second modularity concept that we use is that of feature. A feature can be defined as any self-contained collection of properties and constraints that are recurring within a part or across parts. A window opening, for example, is a self-contained concept with its own properties (e.g. dimensions and angles at corners) and constraints restricting the values of these properties. Further, the concept of window opening is a recurring concept. Doors, roofs, body side panels, and liftgates have window openings. Because features are self-contained concepts, representing them as separate entities contributes to the modularity of the representation. Because the features recur within the same part or across parts, representing them as a separate entity reduces redundancy and promotes modularity and reuse.

3.3.1. Syntax

The concept of part-feature association is an encapsulation concept. For each class, we explicitly list its set of features by naming them (e.g. door handle depression) and identifying their type (e.g. Mho depression). In the following example, illustrated in Figs. 8 and 9, we show the Outer Door Panel class with a feature, a property, and a validation constraint. The rear cutline is 'only' a property because no detailed information is associated with it. The door handle depression, on the other hand, is a feature of type Mho depression with its own properties and constraints.

Features, like properties, are inherited and specialized. Like properties, features are specialized via co-variance. That is, a feature can only be specialized into a feature sub-class of the first one. For example, suppose the front outer door panel has a special type of handle depression, such as FrontHandleDepression, then the Front Outer Door Panel would contain the information shown in Fig. 9.

The implicit requirement in this representation is that the class FrontHandleDepression is a subclass of MhoDepression. The explicit representation of Features also allows the same part/panel to contain many instances of the same feature without needing to repeat the requirements.

3.3.2. Semantics

The part-feature association is a pure encapsulation mechanism. It allows us to represent classes in a modular way, encapsulating details within features and features of features. Within a given class, the features of the class are treated as properties. They are therefore used to define the working space. The properties of the features on the other hand, are encapsulated within the feature class. For example, considering the example of Outer Door Panel shown above in Fig. 8, doorHandleDepression and cutline

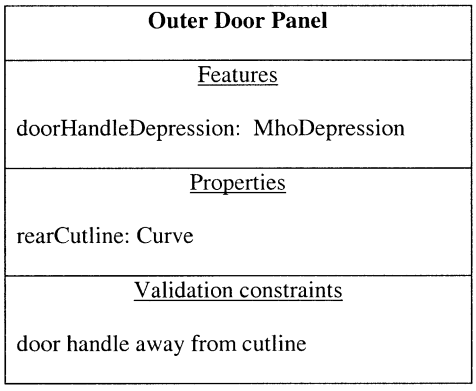


Fig. 8. Part-feature association.

are properties of the working space of the class and can be referred to by the constraints of that class. The depth of the door handle depression on the other hand, is not a property of the working space of the Outer Door Panel class, and cannot be referred to by its constraints.

The distribution of properties and constraints between the part class and the feature classes is illustrated in Figs. 8 and 10. Note that because all of the constraints proper to the feature are encapsulated within the feature class, design decisions on the part proper are not concerned with enforcing validity of the feature anymore than they are concerned with enforcing consistency of a property with its data type. We revise (refine) the definition of class semantics to account for features.

Definition 3.3.1. Definition of class semantics for classes with features. Given a design class D defined by $D = \langle \mathsf{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathsf{F} = \{f_k : F_k\}_{k=1\ldots l}, \mathsf{C} = \{C_j\}_{j=1\ldots k}\rangle$ where P is a set of properties, F a set of features, and C a set of constraints, the class defines a pair of sets $\langle W, T_g\rangle$ as follows:

$$W = T_1 \times T_2 \times \ldots \times T_n \times T_{g_{F1}} \times \ldots \times T_{g_{Fl}}$$

$$T_g = \{s \mid s \in W \text{ and } (C_1 \wedge C_2 \wedge \ldots \wedge C_k)(s)\}.$$

Note that this definition highlights among other things:

- The fact that the part-feature association is not transitive. The feature of a feature is not a feature. Each level of part-feature association introduces a new level of detail/abstraction.
- The fact that the features are fully encapsulated. This is illustrated by the fact that they participate by their target spaces rather than their working spaces. This does not necessarily dictate any specific design approach. Bottom up as well as top down design methods can be applied. The fact that $T_{g_{Fl}}$ figures in W merely emphasizes the fact that the validation of features is outside the jurisdiction of the part's design decisions.

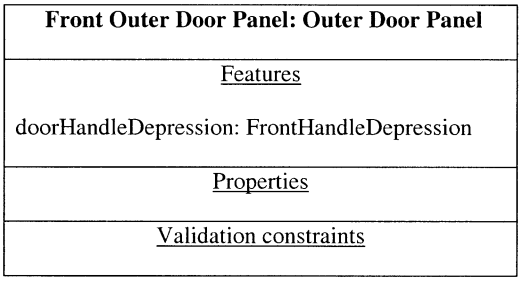


Fig. 9. Feature specialization.

The definition of new features and the specialization of inherited features within sub-classes are regulated in the same way as is the addition and specialization of properties. This is specified by the following axiom.

Feature specialization axiom 3.3.1. Given a design class D defined by $\langle \mathsf{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathsf{F} = \{f_k : F_k\}_{k=1\ldots l}, \mathsf{C} = \{C_j\}_{j=1\ldots k}\rangle$ and given a design class defined by D' : D, $\langle \mathsf{F} = \{f_h : U_h \ldots f_l : U_l, g_{l+1} : U_{l+1} \ldots g_{1'} : U_{l'}\rangle$, the redefined properties $f_i : U_i$ for $i = h$ through l must be such that U_i is a subclass of F_i.

As with sub-classing, the part-feature association should not be cyclic. A part cannot have a feature of the same class as itself nor from its direct chain of inheritance. This requirement carries also for chains of features. We state the two cases individually.

Non-reflexive part-feature axiom 3.3.2. Given a design class D with a feature $f : F$ (native or inherited), F must be different from D, and F must neither be subclass nor a superclass of D.

The rationale of the above invariant is obvious considering the semantics of the part feature association. The working space of the part has the target space of its features as components. This requirement of non-reflexivity is generalized to a chain of part-feature association of any length.

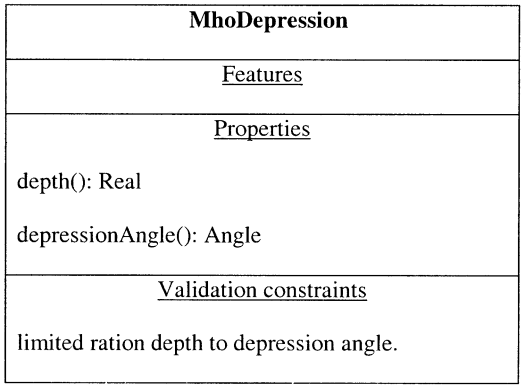


Fig. 10. Feature and its constraint.

Non-cyclic part-feature axiom 3.3.3. Given a sequence of design classes $D_1, D_2, \ldots, D_k$ such that $\forall i: 1 \leq i < k\ D_i$ has a feature of class D_{i+1}, $\forall i, j: 1 \leq i,j \leq k\ D_j \neq D_i$, and D_j and D_i are not be super-classes of each other.

3.4. Assembly-component association

Aggregation is one of the most commonly discussed associations in modeling [28,35]. It is the association between a composite element and its parts. Generally, the components and the composite are independent elements with different sets of properties. Yet, the composite association creates a dependency between *specific* properties of the participant parts and composite. The specific properties that are bound by the association and the manner in which they are bound to each other are domain specific. This contributes to the difficulty of defining general semantics for this association and to the difficulty of providing automatic support to the association by existing languages and systems. In Ref. [17], we define a generic stereotype aggregation that can be customized to individual applications and instances.

In the stamping design knowledge application we are considering, we have a very specialized form of aggregation that plays a predominant role throughout the knowledge base. The association in question is the one relating a part (e.g. door) to the panels composing it (door inner panel, door outer panel). We call this association assembly. Its distinguishing characteristics are as follows:

- It is an association where the composite is a part and the components are panels.
- There are two possible panels composing a part, an inner panel, and an outer panel. Both panels share the shape (cutlines) and some of the features (e.g. openings) as the composite part.
- The panels, i.e. the components are the focus of interest in this application. Properties of the part that are not shared by the panels are of no interest. In other words, the parts are used to capture properties, constraints, and features that are common to both panels of the same part.

In summary, the concept of assembly that we are considering here presents itself in a very simple form, yet it is a predominant concept throughout the book. The predominance of this association makes it requirement to represent it. The relative uniformity and simplicity of this form of assembly make it possible to formalize it. Overall, we consider this experience as a stepping stone towards the formalization of the more general concept.

3.4.1. Syntax

We need to capture the association between a part and the panels that constitute that part. Some parts are composed by an outer panel only (e.g. some roofs), some are composed by an inner panel only (e.g. dashboard panel), and most are composed by both (e.g. doors). In all cases, an assembly is composed by at most two components, and the components are identified by their 'role', inner or outer. This very orderly association is represented by adding a *structure* compartment to design class template as shown in Fig. 11.

The structure component contains the inner panel and outer panel composing the part when the class is a part, and the part of which the class is an assembly, when the class is a component of an assembly (i.e. panel).

3.4.2. Semantics

One of the difficulties with using aggregation operations stems from the fact that aggregates have a mix of properties that are shared by their components and properties that are not. In the assembly association represented here, parts are used as an abstraction for the common properties of panels. The panels of the same part generally share a common outline and some of the features — precisely those features defined by shape such as window opening, wheel opening, pillar, etc. On the other hand, inner and outer panels do have different concerns and different constraints. Outer panels are visible from the outside, and have a number of esthetic constraints ensuring a smooth shiny surface. Constraints on inner panels, by contrast, are more constrained by issues of strength. In summary, an outer door panel inherits some of its features and constraints from its super-class (side outer panel in this instance), but it also 'inherits' from the part door. Note that this is not multiple inheritance. An outer door panel is a side outer panel but it is not a door. It is part of a door.

In light of this, propagation from a part to its panels is very similar to the inheritance propagation. The only difference comes from its discriminative nature. We first illustrate this on an example. We consider the example of the Door class. The Door has a feature window opening of type Opening. Features, like panels, can be specialized into inner and outer. In particular, the set of constraints associated with an opening on an outer panel and that associated with an inner panel are similar but not identical. To be

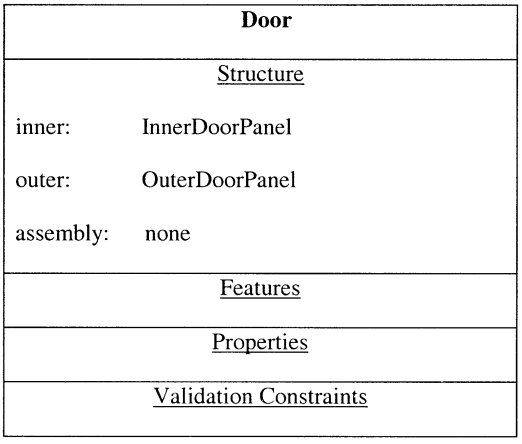


Fig. 11. Assembly representation.

consistent with the trend of reducing redundancy, common constraints are associated with the generic concept of opening, which is then structured as an 'assembly' of an outer opening and an inner opening. Constraints specific to the inner and outer opening are distributed accordingly. On the other hand, an outer door panel has an outer opening, and an inner door panel has an inner opening. This information is captured in a compact way as follows:

- The Class Door is associated with the generic feature opening.
- The Feature opening is structured as an outer opening and an inner opening.
- The Class Door is structured as an Inner Door panel and an Outer Door Panel.

The key here is that the propagation mechanism distributes the information correctly. In other words, the Inner Door Panel does not inherit the overall concept of opening, but its Inner version. The Outer Door Panel will similarly inherit the Outer opening. This propagation scenario is illustrated in Fig. 12. Note that we represent the part-feature association with the bow tie and the assembly association with a circle on the side of the assembly forking towards the components.

This representation with its associated propagation mechanism reduces the quantity of information that needs to be recorded and the level of repetition and redundancy. The representation serves also to enforce symmetry and consistency of the knowledge.

In the example above, we have considered the case where the feature is itself an assembly. If the feature is not an assembly, it is inherited as is by both components of the assembly. We formalize the concept of assembly by defining its semantics and the axioms restricting its use.

The first characteristic of the assembly relationship is that it cannot be nested. A class can be an assembly or a component of an assembly, but not both.

No assembly nesting axiom 3.4.1. Given a design class D defined by $D = \langle \mathit{St} = \langle D_\mathrm{I}, D_\mathrm{O}, D_\mathrm{A}\rangle,\ \mathit{P} = \{P_i : T_i\}_{i=1\ldots n},\ \mathit{F} = \{f_k : F_k\}_{k=1\ldots l},\ \mathit{C} = \{C_j\}_{j=1\ldots k}\rangle$ where D_I, D_O, and D_A are, respectively, the inner component of D, the outer component of D, and the assembly of which D is part. Either D_A is nil or both D_I, and D_O are nil.

With this distinction between classes that are assemblies and classes that are components, we proceed defining them individually. For classes that are assemblies, the components are treated in a way very similar to features and properties. Components as a whole (in their encapsulated form) are part of the working space. Any constraints binding the different components of an assembly are part of the assembly constraints and reflected in the target space.

Definition 3.4.1. Definition of class semantics for assemblies. Given a design class D defined by $D = \langle \mathit{St} = \langle D_\mathrm{I}, D_\mathrm{O}, \mathrm{nil}\rangle, \mathit{P} = \{P_i : T_i\}_{i=1\ldots n}, \mathit{F} = \{f_k : F_k\}_{k=1\ldots l}, \mathit{C} = \{C_j\}_{j=1\ldots k}\rangle$, the inner and outer components of D are treated as components of its space, that is:

$$W = T_1 \times T_2 \times \ldots \times T_n \times T_{g_{F1}} \times \ldots \times T_{g_{Fl}} \times T_{g_{(D_\mathrm{I})}} \times T_{g_{(D_\mathrm{O})}}$$
$$T_\mathrm{g} = \{s \mid s \in W \text{ and } (C_1 \wedge C_2 \wedge \ldots \wedge C_k)(s)\}.$$

On the other hand, inner and outer panels of a part inherit everything (relevant) from their assembly. We specify the inheritance for inner panels.

Definition 3.4.2. Definition of class semantics for inner (outer) components. Given a design class D_I (D_O) defined by $\langle \mathit{St} = \langle \mathrm{nil}, \mathrm{nil}, D\rangle,\ \mathit{P} = \{P_i : T_i\}_{i=1\ldots n},\ \mathit{F} = \{f_k : F_k\}_{k=1\ldots l}, \mathit{C} = \{C_j\}_{j=1\ldots k}\rangle$,

- D_I (D_O) 'inherits' all properties of D.
- D_I (D_O) 'inherits' all features of D that are not assemblies.
- For each feature of D that is an assembly, D_I (D_O) 'inherits' its inner (outer) component if it has one.
- D_I(D_O) 'inherits' all constraints of D.

This inheritance mechanism requires some consistency in the distribution of features to classes according to their nature. For example, an assembly cannot have a feature that is an inner component of another class.

Neutral features for assemblies axiom 3.4.2. Given a

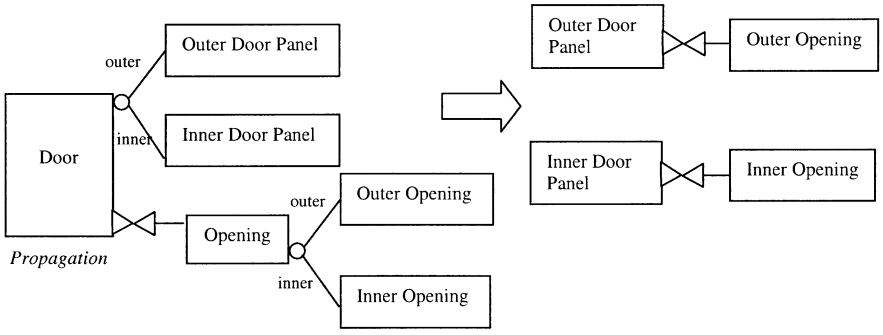


Fig. 12. Assembly and its propagation.

design class D defined by $D = \langle St = \langle D_I, D_O, \text{nil}\rangle, P = \{P_i : T_i\}_{i=1...n}, F = \{f_k : F_k\}_{k=1...l}, C = \{C_j\}_{j=1...k}\rangle$, every feature $f_k : F_k$ must be such F_k is not a component of an assembly.

Component-feature consistency axiom 3.4.3. Given a class D with feature $f_k : F_k$, if D is an inner (outer) component of an assembly, F_k must not be an outer (inner) component of some other assembly.

As with the other associations, reflexivity and cycles are not allowed.

Non-reflexive assembly component axiom 3.4.4. Given a design class D defined by $\langle St = \langle D_I, D_O, \text{nil}\rangle, \ldots\rangle D$, D_I, and D_O must be different, and must not be in a subclass relationship to each other.

4. Summary and discussion

In this paper, we have considered the modeling of knowledge relevant to reflective decision activities during engineering design. We have focussed on the evaluation knowledge. We used an object-oriented model as a means to organize and encapsulate design constraints. Design validation constraints are interpreted as a major component of the specification of engineering objects. These validation constraints are not class invariants. Most alternatives manipulated by designers would not be compliant with these constraints. For the sake of flexibility, and to promote free exploration of the decision space, only final designs are expected to be compliant.

In this paper, we have presented the major components of a knowledge model that we have used to restructure and reorganize information in corporate books of knowledge. The knowledge model was constructed so as to reduce redundancy and preserve consistency, and promote the quality of the information. We have presented here some of the invariants that capture the integrity of the model. Additional invariants have been developed along the model of database normal forms to capture lack of redundancy and economy of expression. Heuristics have been developed to detect violations of these invariants and assist users in repairing the information.

The model was developed to fit a book containing hundreds of stamping requirements. We are in the processing of reviewing the model in light of other books in order to generalize it and adapt it.

Acknowledgements

This project is conducted as a cooperation between DaimlerChrysler Corporation and Oakland University. The first author would like to acknowledge funding from DaimlerChrysler under grant 38330 and Oakland University and the Michigan Research Excellence Fund. We thank the anonymous reviewers for their constructive feedback on this paper and A. Gokani from Oakland University and K. Fry from DaimlerChrysler for their contribution to the project.

References

[1] Ullman DG, Diambrosio B. Taxonomy for classifying engineering decision problems and support systems. Artif Intell Engng Des, Anal Manufacturing 1995;9:427–38.
[2] Shah et al. Research opportunities in engineering design. NSF Strategic Planning Workshop Final Report, April 1996.
[3] Hazelrigg G. Systems engineering: an approach to information-based design. Englewood Cliffs, NJ: Prentice-Hall, 1996.
[4] Brown KN, et al. Constraint unification grammars: specifying languages of parametric designs. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. p. 239–56.
[5] Bernaras A, et al. Problem-oriented and task-oriented models of design in the commonkads framework. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. p. 499–516.
[6] Cleland G, et al. A knowledge-based system approach to the layout design of large made-to-order products. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. p. 257–74.
[7] Gero JS, et al. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1996.
[8] Gero JS et al. A framework for research in design computing. In: Proceedings of ECAADE, 1997.
[9] McMahon CA, et al. Knowledge-based systems for automotive engineering design. In Autotech'91. 1991. pp. 1–7.
[10] Reinschmidt KA, et al. Smarter computer-aided design. IEEE Expert 1994:50–5.
[11] Tong C, et al. Artificial intelligence in engineering design, vol. I. New York: Academic Press, 1991.
[12] Anantha R, et al. Assembly modeling by geometric constraint satisfaction. Comput-Aided Des 1996;28(9):707–22.
[13] Bourne DA et al. Design and manufacturing of sheet metal parts: using features to resolve manufacturability problems. In: Proceedings of the Computers in Engineering Conference and the Engineering database Symposium ASME. 1995. p 745–53.
[14] Chu PC, et al. Feature-based approach for set-up minimization of process design from product design. Comput-Aided Des 1996;28(5):321–32.
[15] Kim G, et al. Design-for-assembly (DFA) by reverse engineering. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. p. 717–34.
[16] McMahon CA, et al. A knowledge-based approach to design for durability in concurrent engineering. J Syst Engng 1994; 1994(1):13–22.
[17] Mili F. Knowledge architecture for engineering design support. In: Proceedings of AAAI spring symposium. 2000.
[18] Narayanan K. Knowledge modeling for engineering design support, Doctoral thesis, Oakland University, Rochester, Michigan, 2000.
[19] Bahler D, et al. An axiomatic approach that supports negotiated resolution of design conflicts in engineering. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. p. 363–79.
[20] Fisher G et al. Seeding evolutionary growth and reseeding: supporting incremental development of design environments. In: Human Factors in Computing Systems (CHI'94). 1994. p. 292–8.
[21] Kamel A. Multiple design: an extension of routine design for generating multiple design alternatives. In: Gero JS, Sudweeks F, editors.

Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994.
[22] Nakakoji K, et al. Perspective-based critiquing: helping designers cope with conflicts among design intentions. In: Gero JS, Sudweeks F, et al., editors. Artificial intelligence in design. Dordrecht: Kluwer Academic Publishers, 1994. pp. 449–66.
[23] Nichols K. Getting engineering changes under control. J Engng Des 1990;1(1).
[24] Umeda Y, et al. Functional reasoning in design. IEEE Expert 1997;March–April:42–8.
[25] Blackler F, et al. Editorial introduction: knowledge workers and contemporary organizations. J Management Stud 1993;30(6):852–62.
[26] Halal WE. The infinite resource: creating and leading the knowledge enterprise. San Francisco, CA: Jossey-Bass, 1998.
[27] Peters R, Özsu M. An axiomatic model of dynamic schema evolution in objectbase systems. ACM Trans Database Syst 1997;22(1):75–114.
[28] Booch G, et al. The unified modeling language user guide. Reading, MA: Addison-Wesley, 1998.
[29] Object Constraint Language (OCL) Specification Version 1.1, Rational Software. 1997.
[30] Bancilhon F, et al. Building an object-oriented database system: the story of O_2. Los Altos, CA: Morgan Kaufman, 1992.
[31] Cardena AF, et al. Research foundations in object-oriented and semantic database system. Englewood Cliffs, NJ: Prentice-Hall, 1990.
[32] Abadi M, Cardelli L. A theory of objects. Berlin: Springer, 1996.
[33] Khoshafian S, Abnous R. Object orientation: concepts, languages, databases, user interfaces. New York: Wiley, 1990.
[34] Booch G. Object oriented development. IEEE Trans Software Engng 1986;12(2):211–21.
[35] Rumbaugh J, et al. Object oriented modeling and design. Englewood Cliffs, NJ: Prentice-Hall, 1994.

Fatma Mili *is associate professor at Oakland University and director of the Software Engineering for Engineering Knowledge (Seek) Laboratory. Research conducted at the laboratory focuses on the software and database support of engineering design activities. She holds a doctorate in computer science from the University Pierre et Marie Curie, Paris, France.*

Wei Shen *is pursuing a master degree in the Software Engineering for Engineering Knowledge Laboratory (Seek), Oakland University, USA. Her current research interests include artificial intelligence, object-oriented database and knowledge-base systems for engineering applications, specifically on semantic modelling, schema evolution and validation for knowledge-base design. She obtained her B.E. degree in Applied Physics from National University of Science and Defence Technology, P.R. China, and her M.S. degree in Physics from Shanghai Institute of Optics and Fine Mechanics, P.R. China.*

Murli Ram *is the Manager of Stamping Technical Systems for Advance Manufacturing Technology group of DaimlerChrysler AG. Mr. Ram's heads a group which is responsible for providing new technology, support and training to Advance Stamping Manufacturing Engineering (ASME) and the Stamping plants. Mr. Ram holds a Bachelors degree in Industrial Engineering from India, along with Masters Degree in Industrial Engineering with focus on computer applications from Ohio University.*

Iliana Martinez *works as a knowledge engineer at the Technical Computing Center at DaimlerChrysler Corporation. She obtained her PhD in 2001, on the topic of development of knowledge-based planning systems. Other areas of research included application of task specific approaches to conceptual design decision making. Her current research interests include the application of knowledge-based approaches to support automotive design.*

Phares A. Noel *is the senior manager for advance manufacturing technology for the North American business group of DaimlerChrysler. He heads the corporate effort for discovering, analyzing and deploying new technologies for the North American and Canadian automotive manufacturing operations. Mr. Noel holds a bachelors degree in electrical engineering and a masters degree in computer engineering from Wayne State University, along with an MBA from the university of Detroit/Mercy.*

Evie Zouras *is a software specialist at DaimlerChrysler. She received a Bachelor of Science in Electrical Engineering with a minor in mathematics from the University of Saint Louis. Her current interest lies in taking software applications designed to organize and capitalize on enterprise wide knowledge, and designing interfaces to present this information in a clear and easily accessible manner.*