Domain Centered Design Support

Fatma Mili and Krish Narayanan
School of Engineering and Computer Science
Oakland University
Rochester, Michigan 48309-4401, USA
e-mail: mili{knarayan}@oakland.edu
Phone: (248)370-2246
Fax: (248)370-4625

Abstract

Design is a creative activity by which artefacts are brought into existence. In engineering domains, this activity is far from being free. It is constrained by a large body of knowledge. In this paper we discuss mechanisms for integrating domain knowledge within design databases in ways that make it most effective for the designer, and most accessible for review and update. In particular, we focus on the aggregation relationship. Aggregation is a very common relationship in design, with very specific properties. Explicit representation of the aggregation relationship enables a whole range of inferences and constraint enforcement by the system.

Keywords

Intelligent CAD systems, Enforcement of semantic constraints, Object-oriented databases, Aggregation relationship, Agile design

1 INTRODUCTION

Design is the ultimate creative activity by which artefacts are brought into existence. In engineering domains, this activity is far from being free. It is constrained by a large body of knowledge. Some of this knowledge is tacit, but some of it is fully articulated and documented. Various efforts in the AI community, the database community, and the CAD/CAM community are converging towards the common goal of integrating domain knowledge into design support tools and systems. The integration would enable the automation of routine aspects of the design, the guidance of the designer towards correct designs, the propagation of changes to designs while preserving compliance with requirements, and the intelligent co-ordination of related design activities.

There are logistic, economic, and technical difficulties to the full and effective integration of domain semantics into design support systems. One such difficulty is the initial investment required in capturing, encoding, and maintaining the domain knowledge. We are interested in design support in those highly regulated engineering domains - such as automotive industry - where the knowledge of interest is already being captured, encoded, and maintained for other purposes. For example, the Society of Automotive Engineers (SAE) publishes a yearly SAE Handbook listing hundreds of requirements, standards, and recommendations relating to automotive design.

In this paper, we present an approach to representing, using, and managing domain information in the context of design support systems. We contend that structure is a central concept in the design of physical objects. A large portion of the domain knowledge is explained and justified directly or indirectly through the objects' structure. Therefore, we seek a data model in which the structural relationship is adequately represented. Furthermore, design activities and contexts are highly dynamic. Design tasks cover a wide spectrum of decisions. Decisions can be well-defined processes such as the materialisation of previously defined structures through parameter setting. At the other end of the spectrum, decisions can involve open-ended tasks involving the definition of new structures in light of new requirements. We are interested in supporting the full range of decisions. The latter kind of decisions requires a representation of the structural relationships with explicit references to the requirements that justify them, and to the behavior that results from them. In section 2, we motivate this research and relate it to other work. In section 3, we present the aggregation relationship as an important concept in design. In section 4, we discuss the representation of domain knowledge relating to the aggregation. In section 5, we discuss the cross-referential knowledge between structure and requirements needed to allow the dynamic modification of structures and of requirements. In section 6, we summarize and conclude.

2 MOTIVATION, RELATED WORK

2.1 Motivation

The needs of design applications, and the inadequacy of the relational model in supporting these needs are by now well known and well documented in the database literature [Katz (1986), Godart (1994), Elmagarmid (1996)]. We briefly review here the main features of interest.

- *Complex data*: The data managed in design systems is complex and highly structured. Traditional first normal form relations with their limited collection of primitive data types are inadequate.

- *More focus on the model*: It is not uncommon for a design transaction to create new types of objects or to mutate existing ones. Traditional database systems are built with the assumption that a database model is stable, defined once, and used as is thereafter. Primitives for evolving the database model are generally limited in their expressive power and restrictive in their usage (e.g. must lock the whole database).
- *Rich semantics*: Data in CAD systems has rich semantics. In business applications, the referential integrity constraint captures rather adequately the semantics of most relationships involved. In design contexts on the other hand, the meaning and consequences of an association between two parts depend on the nature of the relationship, and on the application domain. Such information is the essence of domain knowledge and needs to be an integral part of the database.
- *Cooperative work*: Among other consequences of the complex semantics, is the concept of related transactions. In business applications, two transactions accessing different data items are assumed independent, and left to proceed concurrently; two transactions accessing the same data item are assumed independent but conflicting, and are forced to proceed in sequence - thus isolating them from each other's intermediate results and decisions. In design environments, concurrent transactions are rarely independent. They should not be isolated, they need to be coordinated instead. Transaction management systems need to coordinate tasks based on domain knowledge.

2.2 Related Research and Tools

In this subsection we visit two different areas of database work that directly relate to the issues raised above. The Object Model, as a means to represent complex objects; and active databases, as a means for centrally representing and using domain knowledge.

Object Oriented Model

The relational model of databases with its well-defined semantics, and its industry-wide standards dominated the research and the industry for more than two decades [Stonebraker (1997)]. With the advent and rapid growth of non-traditional applications of databases, the object model is becoming more and more attractive, and is gaining momentum as the main choice for future [Bancilhon (1990), Lamb (1991), Zand (1995)].

The object model has been gaining favor in the database arena thanks to its richer set of built-in concepts [Atkinson(1992), Bancilhon (1990), Catell (1997)]. Objects are the building blocks in an object model. Objects are grouped into classes. A class defines a set of objects with similar states and behaviors. The implementation of a class materializes the states and defines the methods implementing the behavior. The classes are organized into a lattice based on the subclass relationship. Objects in a class inherit state and behavior from the superclasses. The inheritance based on the subclass relationship proves to be a simple but very powerful tool:

- Inheritance allows designers to define classes in an iterative way. The progressive refinement of classes is supported through the creation of specialized subclasses.
- Inheritance provides flexibility and polymorphism by allowing object's membership and behavior to be determined dynamically at run time.
- Inheritance promotes reuse because it supports parametrization, specialization, and redefinition in a natural way.

The data manipulated in a design context lends itself more naturally to an object oriented representation. The classes encapsulate the state of the objects as well as domain specific manipulations of the objects (e.g. move, rotate, show edges, etc.).

Active Database Systems

Object oriented databases allow the representation of complex structures in a natural way, but they do not inherently enforce application-specific structural constraints. Work relating to the expression, management, and enforcement of application specific constraints is grouped under the name of active databases [Widom (1996)]. The concepts introduced in active databases are orthogonal to the data model used. Active rules have been supported to various extents within commercial relational DBMS [Widom (1996), IBM (1995), Owens (1994)], as well as within object DBMS [Widom (1996), Agrawal (1989), Ceri (1997)].

The idea behind active database systems is that there is much more to the definition of a database than the mere specification of its storage data structures. Every application has its own domain-specific knowledge constraining allowable values in the database and allowable manipulations of the data, and dictating operating protocols and mandatory routines. Traditionally, the responsibility of complying with these rules and policies has been delegated to the various application programs through which the database is manipulated. Active database systems advocate the explicit representation of this knowledge as an integral part of the database schema. This knowledge is then centrally managed and used by the database management system. This approach provides for the *knowledge independence* of the application programs. Business rules and constraints may change without affecting application programs. Furthermore, integrity constraints and business rules are made accessible for evaluation, review, and modification.

Active database systems use Event-Condition-Action rules to formulate actions that need to be taken as a reaction to specified events, under stated conditions. For example, a square structure is constrained by the fact that its four sides must be equal at all times. The resulting business rule may consist of automatically propagating a dimensional change of one side, to all remaining sides. This is represented as follows:

Event:	Update(Square.side$_i$, δ)
Condition:	All square's sides are no longer equal
Action:	Forall j in [1,2,3,4], j ~= i do Update(Square.side$_j$, δ) Notify user that update has been propagated to all sides of the square.

Active rules come with their own set of challenges. The crafting of the rules to faithfully express the application's actual constraints and its business rules is not an easy task. It is notably complicated by the potential interaction between rules. Because the actions of one rule can trigger the activation of another, these interactions must be analyzed to ensure the process reaches a quiescent state. Tools and methodologies have been developed to help in the process of designing, analyzing, and updating sets of rules [Ceri (1997)].

Another important feature of rules is their scope. Rules can be specifically targeted to a class, or untargeted. The square rule example is targeted to the class square. The smaller is the scope of a rule, the more efficient is its management.

3 STRUCTURAL RELATIONSHIPS – A GENERIC CONCEPT

3.1 The Power of Genericity

The attraction and decades-long dominance of the relational model is not an accident. The relational model is a one-size-fits-all model that, when it fits, is very easy and convenient to work with. The relational model supports a very small number of concepts with fully defined semantics. Relational database schemas can be defined by simply naming domains, grouping them into relations, and identifying keys and foreign keys. Manipulating relational databases consists of using built-in, declarative languages, and relying on the system to correctly interpret the queries and to optimally schedule them and execute them. Application programs are needed to complement the power of the query languages. Figure 1 contrasts the use of a relational DBMS with that of an object DBMS. In a relational DBMS only a few concepts are supported, but they are supported fully (narrow but deep box relational concepts). An OO system on the other hand, supports a much wider set of concepts (whatever one can express using an OO language), but relies on the user to fully define these concepts. The relational schema is a thin layer built on top of the built-in concepts. Finally, the application programs are also simple calls to built-in, interpreted, and optimized language constructs. The design of an OO database schema is an extensive task requiring the definition of all relevant classes with their methods, their states, and their relationships. Writing application programs consists of accessing these objects through an OO programming language. The flexibility obtained under OO systems does not come for free. It comes with heavy responsibilities assigned to the DBA designer.

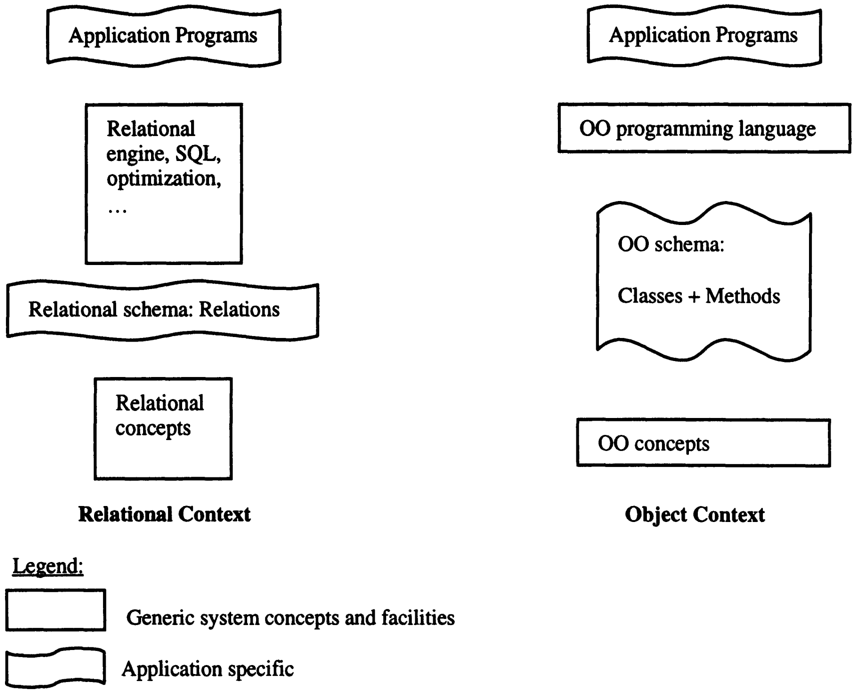

Figure 1 Restrictive-generic vs. Flexible-tailored.

3.2 Structural Relationship

Within the OO model, the only relationship that is universally recognized and supported is the subclass/superclass relationship. With all of its power and interest, this classification relation is mostly a design tool that has little or no visible effects on the end user. Another relationship that has been recognized in the literature [Rumbaugh (1991)] and associated with a graphical notation is the component/part (also called aggregation) relationship. Aggregation is a key relationship in design environments; it embodies most of the essential decisions made during design. To a large extent, designing a system consists of designing (or selecting) its parts and assembling them together. Furthermore, the aggregation relationship is sufficiently interesting in the sense that it has a number of generic properties that are valid regardless of the application specifics. This relationship is - like the subclass relationship - transitive. Whereas inheritance is the universal mechanism for transferring properties through the subclass hierarchy, the transfer of properties from a compound to its part is property-dependent and application-dependent. Properties are *propagated* from a compound to its parts, and from the parts to the aggregate. The propagation mechanism needs to be defined for each instance of the relationship.

3.3 Propagation, Example

To illustrate the propagation of properties from composite objects to their parts, we use an example borrowed from [Mili (1990)]. We consider a communicating vase system, that we call CV. The communicating vase is composed by two vases, a left vase, *V1*, a right vase, *V2*, and a pipe *P* connecting the two. The system CV is schematically shown in Figure 2.

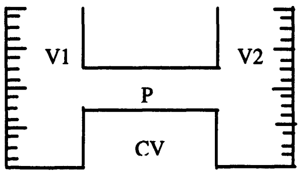

Figure 2 Structure of a communicating vase CV.

Even though *CV*, *V1*, *V2*, and *P* are different objects, of different classes, they are not independent. Moving the vase CV from a location A to a location B will move all of its parts. Filling the vase CV with a fluid will fill the parts accordingly. Similarly, heating or cooling the contents of the vase CV will heat or cool the contents of the parts. The propagation of location, level, and temperature from the vase CV to the vases *V1* and *V2* and to the pipe *P* represent a behavioral relationship that is a direct consequence of the structural relationships. Rumbaugh [Rumbaugh (1991)] acknowledges this general phenomenon, and represents it in the object diagram by an arrow labeled by the propagated operation. In Figure 3, we illustrate the object model of the communicating vase. We show a sample of operations of the class Communicating Vase. Even though all of the operations shown propagate to the aggregated parts, we only show the propagation of the heat and cool operations.

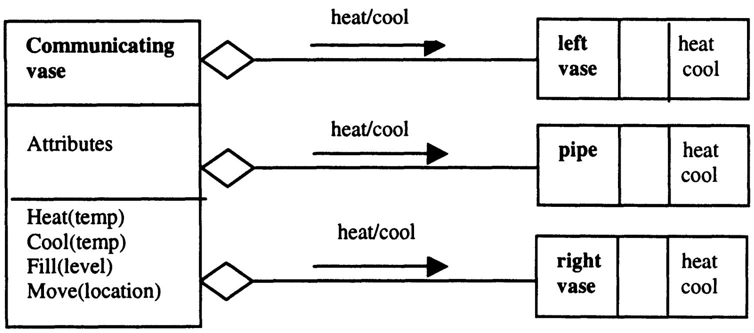

Figure 3 Propagation from compound to components.

The diagrammatic representation of the operations' propagation is only a modelling tool. When the classes are implemented, it is assumed that the containing classes' methods will be implemented properly invoking the operations of the parts as needed. This approach is both unreliable and brittle. It is unreliable because it is dependent on individual programmers and code maintainers' awareness and correct interpretation of the propagation. It is brittle because any change to the structure (e.g. adding a new component) requires major updates to the code. We need a better mechanism for ensuring the proper propagation. So far we have discussed the propagation from the composite object to its parts. The relationship is in fact a two-way relationship: Heating the aggregate heats its parts. Heating any one of the parts raises the temperature of the aggregate.

In sum, we need direct and stable mechanisms for recording structural relationships and their behavioral impact.

4 STRUCTURAL RELATIONSHIPS IMPART BEHAVIORAL RELATIONS

We propose to explicitly represent the structural relationships between composite objects and their parts, and explicitly represent the constraints that result from these relationships.

4.1 Structural Relationships

The object-oriented model provides for the explicit representation of relationships between objects. This representation may be as subtle as having the attributes of one class draw values from the other class, or as explicit as a named relationship with its inverse specified. We propose here to explicitly label aggregation relationships as such. The aggregation relationships in our opinion reflect the essence of objects (e.g. car as it relates to its parts), and thus, should be differentiated from incidental relationships (e.g. owner of a car). The following table summarises the main categories of information we propose to include in the definition of a class.

Table 1 Main categories of information

Class Identification	**Class Name**
Classification Structure	Superclasses, Subclasses
Aggregation Structure	Assemblies, Components
Structural Consistency	Constraints, Associated protocols
Other Constraints	Constraints, Associated protocols
Properties	Attributes, Relationships
Behavior	Operations

These categories together provide a complete description of the class' design, composition, requirements, and behavior. The requirements on the class' contents and behavior are divided between the structural consistency constraints and other constraints. We are focusing here on the structural consistency, and how the structure is reflected in the behavior. We examine the contents of the structural consistency slot in more detail.

4.2 Structural Consistency

We propose to explicitly state the constraints that bind objects and their parts, and complement them with protocols for propagating updates from the assembly to its parts and from the parts to the assembly. For example, in the case of the communicating vases, the information needed includes the following:

Table 2 Information needed

Constraint	*Temperature of whole equals temperature of parts* $\forall x: x \in CV.Components$ $Temperature(x) = Temperature(CV)$
Propagation Protocols	*Propagation from CV to its components* CV.Heat(T) $\Rightarrow \forall x: x \in CV.Components$ do $x.Heat(T)$ CV.Cool(T) $\Rightarrow \forall x: x \in CV.Components$ do $x.Cool(T)$
	Propagation from components to CV x.Heat(T) $\Rightarrow$ if $x \in CV.Components$ do $CV.Heat(f(T,x))$ x.Cool(T) $\Rightarrow$ if $x \in CV.Components$ do $CV.Cool(f(T,x))$

The structural constraint is that the assembly and its parts have the same temperature. This is an equilibrium condition. A change to the temperature of any of the objects must be correctly propagated. The propagation from the composite to the parts here is a straight propagation (cascade). This straight cascade propagation will conceivably be a very common protocol. It is in particular the protocol followed by the shallow copy operation defined for objects. The common definition of shallow copy in fact does not differentiate between incidental relationships (e.g. the vase sits on a table B), and essential relationships such as aggregation. The explicit designation of relationships as aggregation will allow among other things a refinement of the cascading protocol. One can cascade along aggregation only or along all relationships.

The propagation from the components to the whole invokes a function (f) that computes the "averaged" new temperature. An update to the temperature of one component triggers the update of the temperature of the aggregate, which in turn triggers the update of the temperature of all components and reaches a quiescent state.

The information presented here is very similar in contents - although not in form - to the typical ECA active rules. It is intentionally different from the ECA rules in the following respects:

- The condition is stated declaratively, and independently of its enforcement protocols.
- The protocols are listed separately. The protocols can be built-in system defined protocols such as cascade, and restrict, or they can be user-defined protocols.
- The condition and the protocols are represented in an abstract manner. For example, there is no explicit mention to the left vase, right vase and pipe in the example. Instead, the constraint refers to all components of the communicating vase. This approach makes the constraint valid even if the structure changes. Furthermore, the structure provides a rationale for the constraint: It is because the pipe is part of the communicating vase, that its contents are at the same temperature as the contents of the communicating vase.
- As a consequence of the previous point, constraints along with their protocols can be defined high up in the classification hierarchy and be inherited and specialized as needed. General physical and thermodynamic principles can be stated for all aggregates and inherited as is or specialized.

5 ROUTINE DESIGN AND AGILE DESIGN

The framework presented so far incorporates requirements and data together. Object classes are defined in the database schema, spelling out the object's structures, constraints, and behavior. This provides engineers with a prepared environment. The system monitors database events, triggering protocols in response to engineer's actions. Some of the protocols triggered enforce constraints and propagate changes. Not all object constraints can be automatically enforced. The protocols may simply consist of informing the engineer of the violation, explaining it, and possibly proposing a range of actions for correcting it. The ultimate decision belongs to the engineer.

The above scenario is a typical scenario of routine design. Routine design is the design of objects that do not present a novel situation. The objects designed are a refinement (materialization) of the classes defined in the database schema. Routine design is the design of objects whose specifications are not in conflict with the definition of the classes nor with any assumption underlying these classes. For example, the design of a control panel for a specific interior with given dimensions is routine design – unless the dimensions are outside the ranges specified in the class.

There are many situations where design specifications unsettle some of the class assumptions. For example, a communicating vase specification where the pipe is placed close to the top of the vases invalidates the assumption of commutativity and thus the constraint on the temperature. Obviously, the constraint on equal temperatures based on the assumption that the level of fluid is higher than the pipe. Similarly, the design of a control panel for a left-handed driver qualifies as a non-routine design. A large number of standards and requirements on the placement of controls on the control panel are based on the assumption that the driver is a right-handed person. These cases where the specification challenges some of the assumptions underlying the class constraints are called *agile design*. For the system to detect the contradiction, it must be aware of the assumptions underlying classes. Obviously, it would be unrealistic to list all such assumptions, but some obvious ones that we know might be violated, can be included within the class definition. For the specific case of structural constraints, we need to augment the constraint with a set of assumptions on which it is based.

6 SUMMARY, CONCLUSIONS

In this paper, we have presented the main features of our efforts in incorporating, organizing, managing, and using knowledge within the context of design databases. The main characteristics of our approach are as follows:

- Designing around the domain: Traditionally, CAD systems have been designed as generic systems. Domain-specific functionality was then added on top of generic capabilities. We opt here to use an Object-oriented design in which the domain knowledge plays a central role.
- Tightly coupling objects with requirements: We subscribe to the idea that every requirement has a context: The smallest object/aggregate affected by the requirement. Therefore, every constraint is uniquely represented within one class.
- Declarative association between requirements and protocols reinforcing them: Active database systems are mostly concerned with the management of rules resulting from requirements. A change in a requirement has to be manually propagated to all rules enforcing it. We opt instead to represent them together and explicitly relate them. A change to the requirement will automatically flag all associated protocols so that they can be updated accordingly.
- Aggregation: Design data is inherently hierarchical in nature with aggregates composed from other aggregates. We contend that this aggregation relationship has been under-exploited in the OO arena. Making this relationship explicit, and under the full control of the system brings countless benefits.
- Requirements' rationale: Every requirement is based on a set of assumptions. When these assumptions are documented, the system can support in checking consistency and flagging requirements which may no longer hold due to new assumptions.

There are other aspects to this project that have not been discussed in this paper. They include notably the use of structure and requirements for the purpose of concurrency control and cooperation management. The aggregation lattice in particular, proves to be a powerful tool for dynamically decomposing transactions and identifying interference and coordination between related sub-transactions.

7 REFERENCES

Atkinson, M., Dewitt, D., Maier, D. Bancilhon, F., Dittrich, K. and Zdonik, S. (1992) The object-oriented database system manifesto in *Building an object-oriented database system: The story of O_2* (ed. Bancilhon et al.), Morgan-Kaufmann Publishers, Inc., California.

Agrawal, R. and Gehani, N.H. (1989) Ode(object database and environment): the language and the data model. *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, 36-45.

Bancilhon, F. and Dolobel, C. and Kanellakis, P. (ed.) (1992) Building an object-oriented database system: The story of O_2. Morgan-Kaufman Publishers, Inc., California.

Bancilhon, F. and Kim, W. (1990) Object-oriented database systems: In transition. *SIGMOD Record*, **19(4)**, 49-53.

Catell, R.G. and Barry, D. K. (ed.) (1997) The object database standard: ODMG 2.0. Morgan Kaufmann Publishers, Inc., California.

Ceri, S and Fraternali, P. (1997) Designing applications with objects and rules: The IDEA methodology. Addison Wesley Longman, California.

Elmagarmid, A.K, Leu, Y., Mullen, J.G. and Buchres, O.. (1996) Introduction to advanced transaction models, in *Database Transactions for Advanced Applications* (ed. Elmagarmid, Ahmed), Morgan-Kaufmann Publishers, Inc. California.

Godart, C. and Charoy, F. (ed.) (1994) Databases for software engineering. Prentice Hall, New Jersey.

IBM. (1995) IBM DATABASE 2 SQL guide for common servers, Version 2.

Katz, R.H. (1986) Computer-aided design databases in *New directions for database systems* (ed. Ariav, Gad and Clifford, James), Ablex Publishing Co.

Lamb, Ch. (1991) The objectstore database system. *Communications of the ACM*, **34(10)**.

Mili, F. and Mili, H. (1990) The R^2 model: Database = Relations + Relationships, in *Databases in the 1990* (ed. Srinivasan, R. and Zeleznikov, J.), World Scientific Publishing Co.

Owens, K. and Adams, S. (1994) Oracle 7 triggers: Mutating tables? *Database Programming and design*, **7(10)**.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (ed.) (1991) Object-oriented modeling and design, Prentice Hall, Inc., New Jersey.

Stonebraker, M. (1994) Readings in database systems. Morgan-Kaufmann Publishers, Inc. California.

Widom, J. and Ceri, S. (ed.) (1996) Active database systems: triggers and rules for advanced database processing. Morgan Kaufmann Publishers, Inc., California.

Zand, M., Collins, V. and Caviness, D. (1995) A survey of object-oriented databases. *DATABASE Advances*, **26(1)**.

8 BIOGRAPHY

Fatma Mili is an Associate Professor of Computer Science at Oakland University. She has obtained a Doctorate from the University of Pierre et Marie Curie, Paris, France. Her areas of research interests include Decision Support Systems, Software Engineering, and Formal Methods in Computer Science. She is the co-author of two books.

Krish Narayanan is a Doctoral student at Oakland University. She has obtained a Masters in Electronics Engineering from Anna University, India. Her areas of research include Advanced Database Concepts - focussing on active databases, object-oriented databases, and transaction management and Constraint Satisfaction Problems.