Properties of relationships and their representation

by RAMEZ EL-MASRI and GIO WIEDERHOLD
Stanford University
Stanford, California

INTRODUCTION

Data models can provide powerful abstractions to aid in the design of data structures that are relevant to database systems. A large amount of effort has been expended in the development of suitable data models. Several of these models distinguish between classes of entities and relationships among classes of entities in the abstractions they use for modelling the data. Among these models are the network model,1,2 Schmid and Swenson’s model,3 the entity-relationship model,4 Navathe and Schkolnick’s model,5 the semantic data model,5,7 and the structural model.8,9,10

The relational model11 does not explicitly include entity classes and relationships, but concentrates on a simple and uniform representation for all structures as relations. Relationships can then be discovered at query processing time by using relational operators such as the JOIN operator.

The hierarchical model12 represents only 1:N relationships in a straightforward manner, and requires additional reference structures and redundant record types to represent M:N relationships.

We are motivated by two reasons to consider that the correct representation of properties of relationships is important. First, the data model is a representation of some real-world situation, and should reflect as many properties that are known about the real-world situation as possible. Second, most database implementations can take advantage of known relationships among entity classes, and hence it is useful to represent such relationships in the data model. Database processes also take advantage of relationships. Examples include all of the hierarchical and network data languages, but relational languages may also recognize relationships in the database structure. For instance, the relational language SEQUEL 2,3 as used in System-R has a LINK statement to relate tuples from different relations. The language LSL 4 is a general query language for relational, hierarchical, and network data models which utilizes relationships.

All the models cited above, with the exception of the relational model, offer rules that govern existence dependencies of represented entities from different entity classes that are related together by a relationship. The pure relational model does not define such rules, but extensions of the model provide for the definition of arbitrary assertions to state such rules in an integrity subsystem separate from the model itself.15,16 Such integrity constraints are not restricted to relationships, and can define arbitrary rules to govern the behavior of a data model. However, it is useful to explicitly represent the important relationship-oriented constraints that have a natural correspondence to real-world structures, and that can be helpful in designing an implementation of the data model. We hence do not consider tuple constraints as: if sex = male then pregnancies = 0, nor constraints with procedural semantics: managers earn more than their employees. Such constraints will still require separate integrity assertions. We are concerned with constraints of the form: an inventory item must have a supplier.

ENTITY CLASSES AND RELATIONSHIPS

The concept of an entity class is often used in database models.2,4,5,6,10 An entity class is a set of objects of similar structure. For example, an entity class cars-in-California is the set of all cars registered in California, or an entity class car-manufacturers is the set of all car manufacturers.

Each object is described by properties (or attributes) that the object shares with all other objects of the entity class. For example, the entity class cars-in-California may have the properties license-number, color, owner, make, year, while the entity class car-manufacturers may have the properties manufacturer-name, location. Some properties have unique values for each object in the class, and hence may serve to identify each object in the class uniquely. In our example, license-number and manufacturer-name identify uniquely a car and manufacturer respectively.

Clearly, some object may not have all the properties, or may have additional properties that describe them, but all our formal models will impose such a regularity. Not much damage is done: an inappropriate attribute can be given a value not applicable, and exceptional attributes are rarely useful for computational purposes. Subclass concepts can deal with subsets of entities that have additional properties.5,6,10 but this is not directly relevant to our discussion.

A relationship between two entity classes is a mapping that associates with each object of one entity class a number of objects (possibly none) of the other entity class. For example, a relationship car:manufacturer associates with each
car object a related manufacturer object such that the car is made by this manufacturer.

A relationship has rules that govern the mapping of objects from the two entity classes. We shall call these rules the properties of the relationship (different from the use of the word property for entity classes). For example, the car:manufacturer relationship may have the following properties:

1. Every car must be related to exactly one manufacturer.
2. A manufacturer is related to at least one car, but can be related to any number of cars.

These two rules imply that the relationship is a mutual or total dependency: the existence of a car depends upon the existence of its manufacturer, and the existence of a manufacturer depends upon his having manufactured at least one car. The rules also imply that a manufacturer can be related to N cars, when N is unspecified. Hence, the cardinality of the relationship cars:manufacturers is 1:N.

PROPERTIES OF RELATIONSHIPS

There are two important properties of a relationship between two entity classes: the cardinality and the dependency. We first consider the cardinality property. Within each cardinality case, we will discuss the dependency property.

The cardinality property of a relationship places restrictions on the number of objects of one entity class that may be related to an object of the other entity class. The dependency property governs whether an entity can exist independently, or whether it requires the existence of related entities from another entity class. We will see the importance of the cardinality and dependency properties of a relationship when we discuss the representation of relationships. In the ensuing discussion, we will use A and B to denote two classes of entities, and A:B to denote a relationship between the two entity classes.

Cardinalities are classified into three types: 1:1, 1:N, and M:N. Dependencies are classified into four types: total, partial A on B, partial B on A, and no-dependency.

We now discuss the dependency properties as they apply to a relationship of known cardinality.

One-to-one relationships

Consider a 1:1 relationship A:B. We can distinguish four cases based upon the dependency properties of the objects in the two related entity classes.

(a) Total dependency: Every object of class A must be related to one object of class B, and vice versa. In this case, a one-to-one correspondence exists between the objects in the two entity classes.

(b) Partial dependency of A on B: Every object of entity class A must be related to an object of class B, but some objects of class B can exist that are not related to an object of class A. Mathematically, this relationship is defined by a total 1:1 function from A into B. The inverse is a partial 1:1 function from A onto B.

(c) Partial dependency of B on A: This is symmetric to case (b) above.

(d) No-dependency: Objects can exist in both entity classes that are unrelated to an object of the other class. This relationship is defined by two partial 1:1 functions that are inverses of one another.

These properties are relevant in the definition of the semantics of one-to-one relationships. We give examples to illustrate this point.

A marriage relationship between the entity classes husbands and wives is a 1:1 total dependency. A relationship managers:departments is a partial dependency of managers on departments (every manager must be related to a department but short periods of time may exist when a department does not have a manager). Finally, a relationship merchant ships:captains, which describes the current assignment of a captain to a ship is a no-dependency.

One-to-N relationships

Consider a relationship A:B of cardinality 1:N. Here, an object of class B can be related to at most one object of class A, while an object of class A can be related to any number of class B objects. If N is specified as a number, this restricts the number of class B objects related to a class A object to a maximum of N. We can distinguish four types of existence dependencies for such a relationship.

(a) Total dependency (i): Every object of class A must be related to at least i objects, i>0, of class B, and every object of class B must be related to exactly one object of class A. The relationship is defined by a total function from B onto A.

(b) Partial dependency (i) of A on B: Every object of class A must be related to at least i objects, i>0, of class B. An object of class B may be related to at most one object of class A. The relationship is defined by a partial function from B onto A.

Note that we define the dependency of class A objects on class B objects for a 1:N relationship A:B by requiring an object of class A to be related to at least i objects of class B, i>0. We do not define it by requiring an object of class A to be related to exactly i objects, since the latter case can be decomposed into i relationships of cardinality 1:1.

(c) Partial dependency of B on A: Every object of class B must be related to exactly one object of class A. The relationship is defined by a total function from B into A.

(d) No-dependency: Objects can exist in either class that are unrelated to objects of the other class. The relationship is defined by a partial function from B into A.

For examples, see the complete report.17
M-to-N relationships

Finally consider a relationship A:B of cardinality M:N. This is the general case. No restrictions exist on the number of objects related to an object of either class. If M or N or both are specified by numbers, a restriction on the maximum number of objects related to a single object of the other class is specified as in the 1:N case. We can again distinguish four types of existence dependencies for such a relationship.

(a) Total dependency (i,j): An object of class A must be related to at least i objects of class B, i>0 while an object of class B must be related to at least j objects of class A, j>0.

(b) Partial dependency (i) of A on B: An object of class A must be related to at least i objects of class B, i>0.

(c) Partial dependency (j) of B on A: An object of class B must be related to at least j objects of class A, j>0. This case is symmetric to (b).

(d) No-dependency: No restrictions exist on the relationship. This is the most general case.

An example of a partial M:N dependency is the relationship between the two entity classes bills-passed-in-congress and congress-persons that relates each passed bill with the congress-persons who voted yes on the bill. If we assume that a particular congress has 100 members, and a passed bill requires at least 51 yes votes, the relationship bills-passed-in-congress; congress-persons-voting-yes is a partial dependency (i) of bills-passed on congress-persons of cardinality M:N, with i=51.

An example of a no-dependency M:N relationship is that between the entity class suppliers that can supply parts to a company, and the entity class parts that are used by the company. Suppliers can exist that do not supply any parts at some moment, and parts may exist for which there is no longer a supplier. The relationship relates each supplier object with the part objects he currently supplies.

REPRESENTATION OF RELATIONSHIPS OF KNOWN PROPERTIES

A data model is used to represent a real-world situation. The objects in the real-world continuously undergo change. There are many constraints on the way in which the real-world can change. Since a data model implies the specification of rules, it is best if these rules match real-world constraints. We expect representations of objects from entity classes to be inserted into and deleted from the database. These insertions and deletions should be governed by the rules implied in the data model.

A relationship relates objects from entity classes. The properties of relationships discussed in the previous section define rules that govern changes to related objects from two entity classes that have a relationship between them.

These are the rules we wish to consider in this section. When properties of a relationship are known beforehand, it is useful to represent these properties in the model as rules so that updates to the objects that participate in a relationship will maintain these properties. If properties of a relationship are not known, it can be represented in the data model using the most general case: the no-dependency, M:N relationship.

While integrity assertion statements applied to a model are sufficient to describe any rules that constrain the model, they cannot affect the structure of the model, since they are not a part of the data model. As we shall see, known properties of relationships suggest specific model structures, and a small set of rules expressed in the data model can represent all the properties of relationships discussed in the previous section.

Relationships and their constraints in normalized models

An entity class is represented in relation-based models as a set of tuples of identical structure that constitute a relation. Each tuple represents an object from the entity class. The properties of the entity class are described by the attributes of the relation. The attributes define the domains from which data items in a tuple can take values. The values of the identifying, or key, attributes in a tuple define the correspondence between a real-world object and that tuple. Hence, values for key attributes are required to be unique for each tuple in the relation. The non-identifying, or dependent attributes, represent other properties of the object.

In some cases, more than one set of identifying attributes exist. We will assume that one set is designated to identify the objects of the entity class, the primary key (PK), and use that set in our discussion of the representation of relationships.

Relationships can be represented in several ways. The three-relation representation uses a separate relation to represent each entity class. A third association (or relationship) relation, which contains the PK attributes of the other two relations, represents the relationship. Each tuple in this association relation serves to associate tuples from the relations that represent the entity classes.

Figure 1(a) shows an example. Two relations that repre-
sent entity classes managers and departments are related via a third relation manager-dep. We call manager-dep the associating relation, and the attributes emp-no and dep-name in manager-dep the associating attributes. A tuple (55, Payroll) in manager-dep means employee number 55 is the manager of the Payroll department, and associates the tuple with PK 55 in managers with the tuple with PK Payroll in departments. In our diagrams, we will show the PK and the associating attributes in capital letters.

The two-relation representation includes the associating attributes in one of the relations that represent an entity class. Figure 1(b) shows an example, using the same manager:department relationship. This representation only works for relationships of cardinality 1:1 or 1:N, since only one value of the associating attribute (the dept-managed attribute in our example) can exist in a normalized relation.

For 1:1 relationships, a one-relation representation is possible (Figure 1(c)). It is also possible to represent the associating attributes twice, once via the associating attribute dept-managed in managers and a second time via the associating attribute managers in departments. One of the associating attributes is redundant, and to assure that a manager tuple is associated with the same department tuple in both representations, an additional constraint is needed.

Since a large number of integrity constraints can be costly to manage, one measure of goodness for the representation of a relationship is that it requires a minimum of additional constraints, but still correctly represents the properties of the relationship.

When tuples that participate in a relationship are inserted or deleted, the transformation must not lead the database from a consistent state into an inconsistent one. The properties of the relationship define conditions for a consistent state. Hence, update constraints that maintain the properties of a relationship should be specified in the data model. We can consider updates of the database as transactions which transform one consistent state of the database into another.

Update constraints are specified on the relations of the data model to reflect the properties of the relationship. We now discuss these constraints for representations of relationships between two entity classes.

The associating attributes (AA) constraint, and rules to maintain it

The associating attributes constraint ensures that we are associating two existing tuples (that represent two exiting entities) in the database: it does not make sense to associate a department which is not in the database with a manager. Hence, this AA constraint specifies that the values of associating attributes must correspond to values of primary keys in the relations that represent the two related entity classes.

The consequences of this constraint for insertion and deletion of tuples are:

1. Insertion of a tuple in a relation with associating attributes is permitted only if the values of the associating attributes correspond to values of primary keys of tuples in the relations that represent the entity classes.
2. Deletion of a tuple from a relation that represents an entity class is permitted only if the value of the primary key does not correspond to a value of associating attributes in some tuples that remain in the database after the deletion.

The insertion rule is straightforward. Attempts to insert a tuple which violates the rule are rejected. To remedy the situation, one must first insert other tuples that make the required insertion legal. Another possibility is to insert this tuple, and other tuples associated with it simultaneously as a transaction so that after all tuples in the transaction are inserted, the insertion rule holds. The whole transaction may be rejected if it results in a violation.

For deletion, two rules can be specified to ensure the consistency. The prohibit deletion (or PD) rule aborts the deletion until the tuples with associating values that correspond to the PK of the tuple being deleted are explicitly deleted. The delete related tuple (or DLT) rule carries out the deletion, and automatically deletes all tuples with associating values that correspond to the PK of the tuple being deleted to ensure the consistency. This may result in the deletion of tuples that represent objects from the other entity class.

The cardinality constraint

We now consider how cardinality constraints can be expressed in a data model. Consider the three-relation representation. If we specify no cardinality constraints, it represents a general M:N relationship (Figure 2(a)). For a 1:N relationship if we restrict the associating attribute emp-no in the relation dept-emp to have unique values in the different tuples of dept-emp at all times, the 1:N cardinality is guaranteed. To further restrict the cardinality to 1:1, we specify that both associating attributes have unique values also (Figure 2(c)). In our diagrams, we specify this uniqueness constraint by a (U).

The two-relation representation can represent a 1:N or a 1:1 cardinality. Figure 3(a) shows a 1:N relationship. If we restrict the associating attribute to unique values (Figure 3(b)) we get a 1:1 cardinality for the relationship. Note that the UA constraint can also be used to specify the PK when it is a single identifying attribute of a relation that represent entity classes (Figure 2 and 3).

The dependency constraints

Dependency constraints of a relationship A:B specify whether an object from one of the entity classes can exist independently, or whether the object must be related to objects from the other class at all times.

A no-dependency relationship is best represented by the three-relation representation. Tuples then exist independ-
To maintain such a constraint, insertion of a new department tuple must specify at least \( i \) employee tuples that already exist in employees, and a transaction insert-new-department will insert the new department and the least \( i \) associating tuples in dept-emp. Insertion of an employee is unconstrained, since it will never cause an inconsistency.

Deletion of a department or employee tuple can be governed by either the PD or the DLT rules. Deletion of an associating dept-emp tuple can result in an inconsistency if it reduces the number of employees tuples associated with a department tuple to less than \( i \). We can either specify the PD rule (prohibit deletion of the associating tuple and do not carry out the requested deletion transaction) or the DLT rule (delete the department tuple, and all tuples in dept-emp associated with it). In this case, the PD rule is more plausible, and can affect that the enterprise reconsider the state of its departmental structure.

A partial dependency of a relationship of cardinality \( M:N \) can be handled in an identical way to the \( 1:N \) partial dependency. The only difference is the absence of the \( (U) \) specifying the cardinality on the associating attribute emp-no in dept-emp. Total dependencies are handled similarly, but require two rules for maintaining the consistency.

**Non-normalized relations**

If we allow relations that are not in first normal form\(^{11}\) a relation may include a repeating attribute. A repeating attribute allows a set of values for the attribute in a tuple of the relation rather than a single value. Now \( 1:N \) and \( M:N \) relationships with partial or total dependency can be represented more naturally using the two-relation representation. This is because we can allow the associating attribute to associate a single tuple to a set of tuples of the other relation.

Consider the partial dependency \( (i) \) of departments on employees in a \( 1:N \) relationship departments:employees. It can be represented as in Figure 5(a). The associating attribute emp-no in departments is a repeating attribute, and the \( (i) \) means at least \( i \) values for this attribute must exist in each tuple of the departments relation. The \( (U) \) specifies here that values of the attribute in all the relation are unique, and hence ensures the \( 1:N \) cardinality.

A partial dependency relationship of cardinality \( M:N \) can be represented similarly (Figure 5(b)) with the \( (U) \) removed to allow the \( M:N \) cardinality.

**COMPARISON OF THE REPRESENTATION OF RELATIONSHIPS IN MODELLING APPROACHES**

In this section, we compare the representation of relationships in the network model, the entity-relationship model, Schmid and Swenson’s model, Navathe and Schkol-
From the collection of the Computer History Museum (www.computerhistory.org)

The network model

The network model uses record types to represent entity classes, and link-sets to represent 1:N relationships. To represent an M:N relationship between two record types, two link-sets and an additional link record type are defined. We do not include the basic relational model since it does not explicitly represent relationships in the model, but requires integrity assertions separate from the model to represent relationship constraints. While this is completely general, it does not influence the design of the data model, and may be expensive to maintain in an implementation.

The entity-relationship model

The entity-relationship model uses entity relations to represent entity classes, and relationship relations to represent relationships. The entity-relationship diagram can specify the cardinality of a relationship explicitly by stating whether it is 1:1, 1:N, or M:N. Two types of entity relations are defined: regular entity relations and weak entity relations. A no-dependency relationship of any cardinality is directly specified by a relationship relation between any two regular entity relations. A partial dependency of B on A in a 1:N relationship A:B is specified by a weak relationship, where entity relation A is a regular entity relation and entity relation B is a weak entity relation. The DLT rule is used to maintain the consistency in both cases.

Schmid and Swenson’s model

Schmid and Swenson’s model uses third normal form relations. Entity classes are represented by independent object types, and relationships are represented by association relations. There are three other types of relations in the model, but we only consider independent object type and association relations, since they are used to represent relationships between entity classes. An association represents a general M:N no-dependency relationship. The AA constraint is maintained by the PD rule. Hence, associating tuples must be explicitly deleted before deleting the independent object type tuple. Representation of 1:1 and 1:N relationships require additional constraints, separate from the model, as do dependency constraints.

Navathe and Schkolnick’s model

Navathe and Schkolnick’s model uses entities to represent entity classes and simple associations to represent relationships. Other types of associations exist. Since we are here only considering relationships between different entity classes, and not subclass relationships, we will only consider simple associations. Two type of connectors exist in this model: directed and undirected.

We will categorize the three examples of simple associations given in A simple association between two entities A and B, where the connectors are not directed, defines a no-dependency M:N relationship A:B. Consistency is maintained by the DLT rule (deletion of association tuples when an entity is deleted).

An owner-member simple association, with directed connectors from the owner entity A to the association, and from the association to the member entity B, defines a 1:N relationship A:B. The dependency is a partial dependency of B on A, since tuples cannot exist in B that are not related to tuples in A. The consistency is maintained by the DLT rule.

A simple association with an undirected connector from entity A to the association, and a directed connector from the association to entity B, specifies an M:N relationship.
A:B that is a partial dependency (1) of B on A, since each entity in class B must be related to at least one entity of class A. To maintain the consistency, when an entity of class A is deleted that is the only entity related to a particular entity of class B, this related class B entity is also deleted, which is the DLT rule.

Additional integrity assertions, separate from the model, may be needed to specify other cardinalities and dependencies.

The semantic data model

The semantic data model\(^7\) is a rich semantic model which supports powerful user interface facilities. It provides concrete object classes to support entity classes, and inter-class connections and event class to represent relationships. Many semantic constructs are part of the model, including subclasses, abstractions, aggregates, and events. Detailed specification is provided for attributes, such as mandatory (null values not allowed), unique, and ordering of values. Multi-valued attributes are permitted.

We will discuss here what types of relationships inter-class connections can support. Suppose a relationship A:B is represented by an attribute in the concrete object class A that references objects from concrete object class B. If the attribute is specified to be single-valued and unique, the relationship is 1:1, and if it is specified to be single valued and non-unique, the relationships is N:1. If the attribute is a set (repeating attribute) and unique, the relationship is 1:N, and if it is a set but not unique, the relationship is M:N.

For single-valued attributes that are mandatory (no null or unknown value), the relationship is a partial dependency of A on B for 1:1 or N:1. For a mandatory set attribute, a partial dependency (1) is specified for 1:N or M:N. Total dependencies and partial dependencies (1) require additional constraints, separate from the model.

The consistency of the AA constraint is maintained using the PD rule, since objects in class B that are referenced by objects from class A may not be deleted.\(^7\)

The structural model

The structural model\(^9\) is formed from relations in Boyce-Codd normal form and three types of connections: ownership, reference, and identity connections. Identity connections are used to represent subrelations of existing relations, and carry out functions similar to the categorization, selection, and subsetting associations of Navathe and Schkolnick's model, and the subclasses of the semantic data model, so we will not consider them further. Sets of attributes can be defined to have unique values, using the (U) notation described above.

A partial dependency of B on A of cardinality 1:N is specified either using a reference connection (Figure 6(a)) or an ownership connection (Figure 6(b)). We show the 1:N relationship departments:employees with partial dependency of employees on departments. The reference connection maintains the AA consistency using the PD rule, while the ownership connection maintains it using the DLT rule. Both connections associate a department tuple with N employee tuples, and hence represent a 1:N relationship. The 1:1 relationship can be specified by restricting the associating attribute dept in the employees relation to unique values.

No-dependency relationships are represented using three relations and two connections. A 1:1 or 1:N cardinality is specified by restricting the associating attributes to unique values, otherwise the relationship is M:N. Figure 7 shows the 1:N no-dependency relationship department:employees. An ownership or reference can be used to specify DLT or PD for each entity class.

By choice of ownership or reference connections, the consistency maintenance rules are specified. In Figure 7(a),

![Figure 7](https://example.com/figure7.png)

Figure 7—No-dependencies in the structural model.
deletion is unrestricted for both employee and department tuples, and the AA consistency maintained using the DLT rule. In Figure 7(b), deletion is unrestricted for employees (DLT), and restricted for departments associated with at least one employee (PD). In Figure 7(c), the PD rule is used for both department and employee tuples. The choice of representation depends upon whether a check is desired when tuples that are related to other entity classes are deleted.

Total dependencies, and partial dependencies (i) for 1:N and M:N relationships, cannot be represented in the structural model.

**SUMMARY AND POSSIBLE EXTENSIONS**

None of the above models allows a full choice for representation of the properties of relationships discussed. In particular, partial dependency (i), i>1, for 1:N and M:N relationships, and total dependency relationships, cannot be represented without the specification of constraints separate from the model.

Table 1 shows what relationship properties each of the six modelling approaches can represent without additional integrity constraints.

An extension to the structural model can allow representation of all the relationship properties discussed earlier. We can suggest similar extensions for other models, and perhaps some of them have already been considered by their authors. The structural model is the one we are most familiar with.

We attach a constant number <1> to an ownership or a reference connection. This means at least 1 tuples from the relation at the N side of the connection are attached to each tuple at the 1 side of the connection at all times. If two numbers <1→2> are attached to the connection, a minimum of 1 and maximum of 2 tuples are allowed. The default is <0, infinity>. Now, all cardinalities and dependencies discussed can be represented.

**CONCLUSIONS**

In this paper, we examined the semantic properties of a relationship between two entity classes. We identified two properties: the cardinality and the dependency, and discussed the different cases of each property.

We then showed how these properties can be represented using normalized relations, and what update constraints are necessary to maintain these properties. We identified the basic AA (associating attribute) constraint and described two rules to maintain it: DLT (delete related tuples) and PD (prohibit deletion). We then discussed the cardinality and dependency constraints.

We compared six data models with respect to the choices they offer for the representation of relationships. We subsequently showed how extensions can represent all properties of relationships discussed. We suggest that the following dimensions be used in describing relationships in data models:

1. (1) Cardinality: 1:1, 1:N, or M:N
   (2) Dependency: Total, partial (2), no-dependency.
   (3) Deletion rule to maintain consistency: DLT or PD

**ACKNOWLEDGMENT**

This work was supported by the Defense Advanced Research Projects Agency, Contract MDA 903-77-C-0322, under the KBMS Project.

**REFERENCES**

Earth Resources

The purpose of this session is to present a cross-section of talks relating to the current state of computer simulation modeling as applied to solar energy systems. This field is of major importance today because of the national need to explore alternative energy sources in a quick, comprehensive, and inexpensive manner.

Since a large scale solar energy industry does not presently exist, the application of these models can save years of time that might otherwise be wasted on unfeasible systems. The models to be discussed deal with:

- Engineering and economic performance of solar thermal and photovoltaic power plants.
- The effects of ownership options, government policies and operating alternatives on the economic viability of small photovoltaic systems.
- The interaction of solar electric power plants with existing utility grids.

Leigh S. Rosenberg
Area Director