Generic Relationships for Better Conceptual Modeling

Manuel Kolp

University of Toronto
Faculty of Information Studies

February, 12, 2001

Relationships for Better Conceptual Modeling

• Relationships
• Status of Relationships in Databases and Information Systems
• Problems
• Interest of Generic Relationships
• Generic Relationship Semantics
• Examples: Aggregation and Materialization
• Full Fledged Relationships in Systems
• Metaclass Implementation
• Conclusion
Modeling Relationships

- **Class Level**: A relationship type $R$ among entity types $E_1, \ldots, E_n$ models a structural association between real-world entities.
- **Instance Level**: Relationship instances $r_i = (e_1, \ldots, e_n)$ of $R$, where each $e_i \in E_i$, model links between individual entities (instances of the real-world association).

Status of Relationships in Database and Systems

- Typical model of languages (e.g., C++, Smalltalk, Java) and commercial DBMSs (e.g., ODMG systems) is **very basic**.
- Examples: Support **low-level** version of:
  - generalization: basic built-in class/subclass, or inheritance mechanisms
  - aggregation: basic link expressed by inter-object references as pointer-valued attributes
- More specific relationship semantics directly supported **neither by syntax nor by semantics**
- Relationships can only be represented **indirectly**, in terms of other model construct
Example: Pointer-Based Approach

class Employee {
friend Department;
char * SSN;
Department * employer;
public:
Employee(char* aSSN);
~Employee();
void join_dept(Department *);
}

class Department {
friend Employee;
char * name;
Set<Employee *> employs;
public:
Department(char* aName);
~Department();
void add_employee(Employee *);
void delete_employee(Employee *);
}

- Relationship implemented by two attributes
  (employer from each Employee to a Department and employs from each Department to a set of Employees)
- No Specific Relationship Semantics, Construct

Object Database Management Group Relationships

- Binary relationships only
- Between objects only, literals cannot participate in relationships
- Not “first class” (they are not objects, they don’t have an identifier)
- Implicitly defined by declaring traversal paths in pairs

interface Professor {
...
relationship set<Course> teaches
inverse Course::is_taught_by;
}
interface Course {
...
relationship Professor is_taught_by
inverse Professor::teaches;
}
Generic Relationships

- **Generic relationship** = abstract template (metarelationship) from which concrete specific relationships are derived
- Well-known generic relationships are
  - generalization (Superclass ← Subclass)
  - classification (Class <- -Instance)
  - aggregation (Whole ◊—Part)
- **Specific relationship** = a realization of generic relationship in a particular application (e.g. Article—◊Journal)

Conceptual Modeling: Narrowing the Semantic Gap

- **Goal**: improve models and tools for representing information and processes
- **Why**: narrow the gap between concepts in the real world and their representation in conceptual models
- **How**: identify and formalize powerful modeling primitives, like generic relationships, for more accurate and intuitive descriptions of real-world concepts
Conceptual Modeling in the Database Design Process

- Real world
  - Requirements
    - Collection and Analysis
    - Database Requirements
    - Conceptual Design
      - Conceptual Schema (in a high-level data model)
        - DBMS-independent
        - Logical Design (Data Model Mapping)
          - Logical Schema (in the data model of the DBMS)
            - Physical Design
              - Internal Schema

Some Generic Relationships

- **Classification**: an instance to its class (e.g., John and person)
- **Generalization**: a superclass to subclasses (e.g., person and employee)
- **Aggregation**: composites (e.g., car) formed from components (e.g., body and engine)
- **Materialization**: a class of categories (e.g., models of cars) and a class of more concrete objects (e.g., individual cars)
- **Ownership**: an owner class (e.g., persons) and a property owned (e.g., cars)
- **Grouping**: a member class (e.g., players in a team) to a grouping class (e.g., teams)
- **Viewpoint**: partial information about a class from a particular standpoint
- **Generation**: new output entities emerging from input entities
- **Versioning**: an object class and its time-varying versions
Semantics of Generic Relationships

- **Class and instance semantics**: The semantics of GRs must coordinate both the *class (entity) level* and the *instance level*. (E.g., for generalization: a class can have several superclasses and several subclasses; an instance of a class is also an instance of all its superclasses)

- **Cardinality**: the number of objects related by a relationship (E.g., for materializations: AbstractClass (0,n) → ConcreteClass (1,1))

- **Composition**: A class plays several roles of the same GR R in several specific relationships based on R. (E.g., Car ⊓ Body ⊓ Door)

- **Transitivity**: can follow from composition. E.g., Person ⇐ Student ⇐ GraduateStudent imply Person ⇐ GraduateStudent.

- **Multiplicity**: the same class can participate with that role in several realizations of the GR. (E.g., a composite with several components)

- **Exclusiveness/sharing**: An object can be shared among other objects

- **Existence dependency**: An object exists dependently of related objects

- **Attribute inheritance and value propagation**: propagating structure and behavior from one participant to another.

---

Aggregation in our General Model of Relationship

- **Semantic relationship** from real world modeling part relationship with specific semantics to characterize various dependencies between parts, or *component objects*, and aggregate, or *composite objects*
Materialization

- Binary Relationship between an abstract class and a concrete class
- CarModel is more abstract than Car
- Flavor of is-a generalization / Flavor of is-of classification
- Cardinalities: normally (0,n) and (1,1)
- Inheritance (or attribute propagation)

```
<table>
<thead>
<tr>
<th>AC : CarModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>name (T1)</td>
</tr>
<tr>
<td>sticker_price (T1)</td>
</tr>
<tr>
<td>eng_size (T2, mono)</td>
</tr>
<tr>
<td>auto_sound (T2, multi)</td>
</tr>
<tr>
<td>special_equip (T3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CC : Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>manuf_date</td>
</tr>
<tr>
<td>serial_nb</td>
</tr>
<tr>
<td>owner</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FiatRetro_Cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Fiat_retro</td>
</tr>
<tr>
<td>sticker_price = 10 000</td>
</tr>
<tr>
<td>eng_size = 1200</td>
</tr>
<tr>
<td>auto_sound = (tape, radio)</td>
</tr>
<tr>
<td>alarm = Acme</td>
</tr>
<tr>
<td>airbag = Burglar_King</td>
</tr>
<tr>
<td>cruise = Fiat</td>
</tr>
<tr>
<td>manuf_date = 1/1/96</td>
</tr>
<tr>
<td>serial_nb = 123087657</td>
</tr>
<tr>
<td>owner = Nico</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FiatRetro</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Fiat_retro</td>
</tr>
<tr>
<td>sticker_price = 10 000</td>
</tr>
<tr>
<td>eng_size = (1200, 1300)</td>
</tr>
<tr>
<td>auto_sound = P(tape, radio)</td>
</tr>
<tr>
<td>alarm = string</td>
</tr>
<tr>
<td>airbag = string</td>
</tr>
<tr>
<td>cruise = string</td>
</tr>
</tbody>
</table>

A Two-faceted construct
```

Full-Fledged Relationships in Systems

- Storing all relevant semantics
- Leads to integrate relationship as a language primitive
  → Reifying Relationships
- Keep, propagate the real-world semantics of generic relationships identified in upstream phases in downstream phases
- To do that → Metalevel mechanisms
Computational Reflection and Metalevel

- Ability of a system to observe and manipulate as data the state and behavior of the system itself during its own execution.
- Two-level architecture:
  - Base level: ordinary programming level.
  - Metalevel: where the base level is represented (reflected).
- Causal connection: any change in the metalevel → corresponding effect upon base level.

![Diagram showing the metalevel and base level connections](image)

Metalevel, Reflection and Causal Connection

![Escher’s drawing hands](image)
Structural Reflection

- Meta2class Level
  - Meta2class

- Metaclass Level
  - Entity

- Class Level
  - Person

- Object Level
  - John

Behavioral Reflection

- Metaobject classes
- Metalevel
- Base Level

- Metaobjects
- Base level classes
- Base level objects

MetaAttribute
MetaMethod ()
Implementing Part Relationship with Metaclasses

- **ConceptBase**: Deductive object manager system for meta databases intended for conceptual modeling.
- **Telos** model, includes **Metaclasses** (e.g., Class, Attributes, Proposition)

```plaintext
Cpn.Class in Class with attribute composite: Cps.Class end
Cps.Class in Class with attribute component: Cpn.Class end

Cpn.Class with rule cpnDedRule:
(∀ Cps/Cpn.Class Cpn/Cpn.Class)
(Cpn composite Cps) ⇒ (Cps component Cpn)
```

- **Structural Reflection**
  - Two metaclasses, Cpn.Class and Cps.Class, instances of Class
  - Meta-attributes composite and component: semantics of the PR in each direction.

- **Behavioral Reflection**
  - Rule cpnDedRule for Cpn.Class specifies the automatic creation of a composite-to-component link upon creation of a component-to-composite link
Conclusion

- More powerful conceptual models improve database and system design
- → Relationships
- Contributions
  - Relationship Issues in Information and Database Systems
  - A generic relationship model
  - → Extends current methodologies for better modeling (concepts, data, information, knowledge)
  - A reflective architecture for relationship reification
  - → Enhances Database/Information/Knowledge Systems Expressiveness