Specification Notations

CSE 403 - Spring 1997

Outline

- The uses of specifications and their qualities
- Why specification notations?
- Operational Specifications
  - Data Flow Diagrams
  - Finite State Machines
  - Petri Nets
  - State Chart
- Descriptive Specifications
  - Entity-relationship Diagrams
  - Logic Specifications
  - Algebraic Specifications

The Uses of Specifications

- A statement of customer needs.
- A statement of the requirements for the implementation.
- A statement of the organization and architecture of the implementation.
- A reference point during product maintenance.

An artifact to establish a clear mutual understanding and communication among the parties involved in a software project.

Specification Qualities

- Clear, unambiguous, understandable.
  - Assumptions?
- Consistent
- Incrementality
  - Maintainability/evolvability.
- Traceability.

Why Specification Notations.

- A specification notation will allow you to state certain aspects concretely.
  - The process of posit your requirements/design, etc. in terms of notations would force you to think!
  - Akin to writing natural language prose.
  - Make it easier to answer questions on certain aspects of requirements, design, etc.
- A formal specification notation may allow simulation and some analysis of property.
- Operational vs. descriptive specifications.
  - Operational specification → desired behavior.
  - Descriptive specification → desired properties.

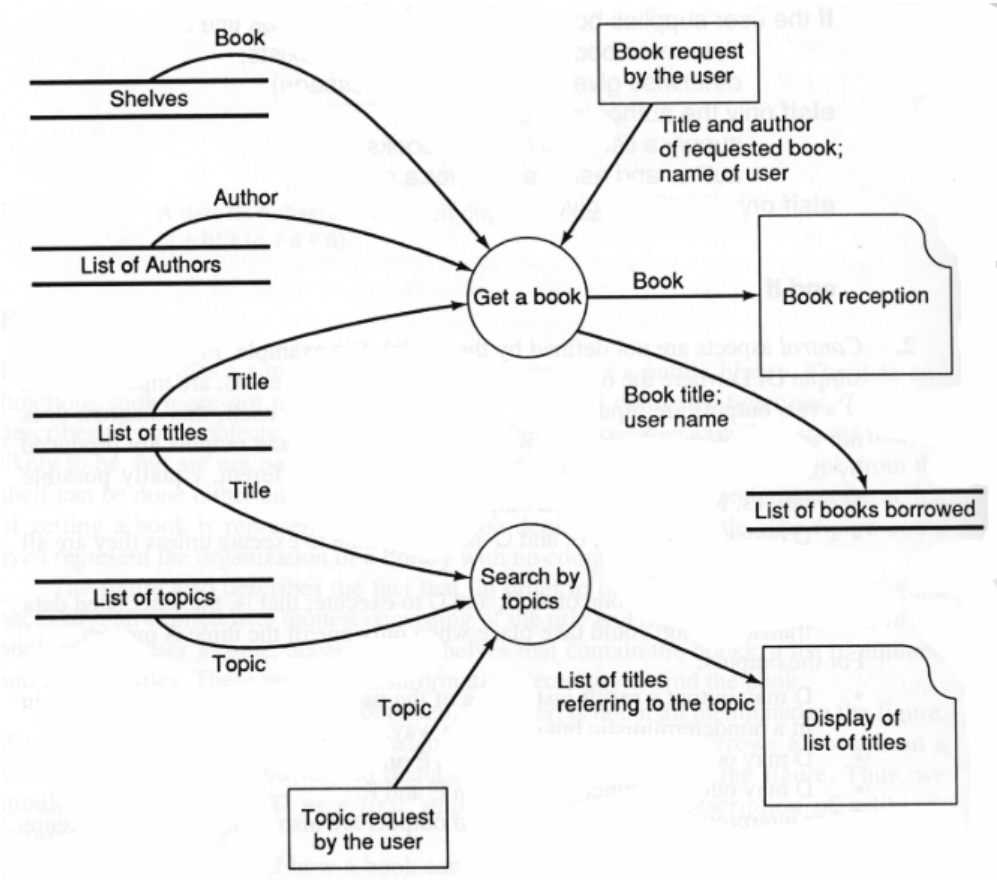
Dealing with Weaknesses of a Notation

- No single notation can capture all aspects.
- What to do?
  - Use a complementary notation.
  - Augment the notation.
  - If the the definition of a notation is not formal, revise it to make it formal.

Data Flow Diagrams [GJM]

- Components
  - Bubbles → Functions/processing
  - Arrows → Data flow.
  - Open boxes → Data store.
  - I/O Boxes → Input/output, interaction with the external environment.
- Informal semantics.
- Control aspects are not defined by the model.

Data Flow Diagram Example [GJM]



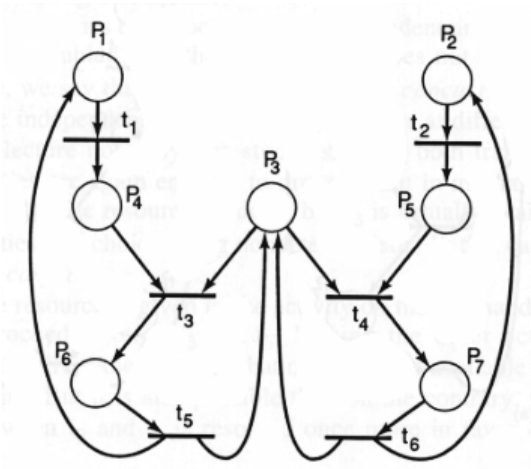
Finite State Machines

- Describes control aspects formally as transitions between states on inputs.
- I.e.,
  - A finite set of states, Q;
  - A finite set of inputs, Σ;
  - A transition function δ: $Q \times \Sigma \rightarrow Q$.
  - An initial state, $q_0 \in Q$.
  - A set of accepting state, $F \in 2^Q$.
- Weaknesses
  - Limited expressiveness.
  - Not suited to express concurrent systems.
- Remedies?

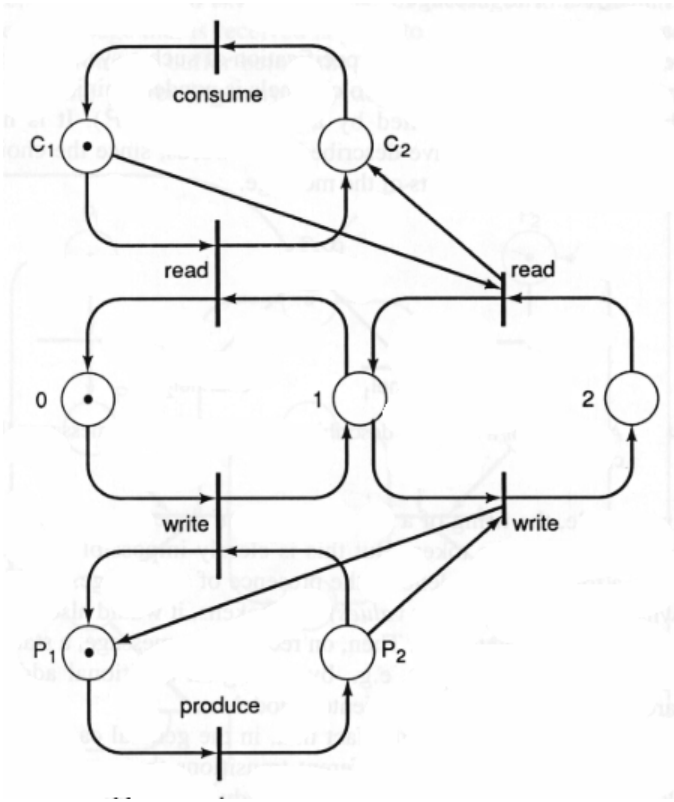
Petri Nets

- A graphical formalism for specifying (asynchronous) systems.
  - Allows reasoning about certain properties of a system, e.g., concurrency, starvation, deadlock.
- I.e.,
  - A finite set of *places* (*circles*).
  - A finite set of *transitions* (*arrows*).
  - A finite set of *arrows* connecting either places to transitions or transitions to places.
- A Petri net is given a state by marking its places.
  - Graphically, represented by *tokens* in places.

Petri Net Example



A Petri Net for a Producer-consumer System



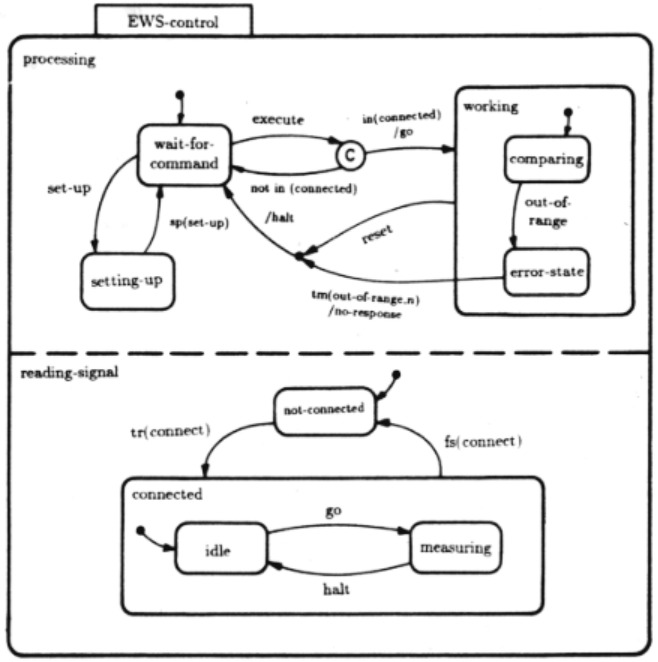
Limitations of Petri Nets

- Expressing a selection policy.
- Timing issues.
- Expressing a scheduling policy.
- Lack of hierarchical decomposition.

Statechart [Harel]

- State based notation.
- Transitions are guarded and with actions.
  - e[C]/a
- Hierarchical decomposition.
  - OR- and AND-decompostions.

Statechart [Harel]

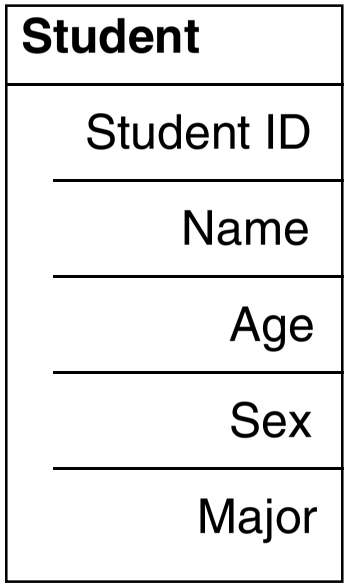


Entity-relationship (ER) Diagrams

- A conceptual description of the structure of data and of their relations.
- ER model consists of:
  - *Entities*
  - *Attributes*
  - *Relations* - **binary** relations.
- ER diagram describes an ER model graphically.

Components of an ER Diagram

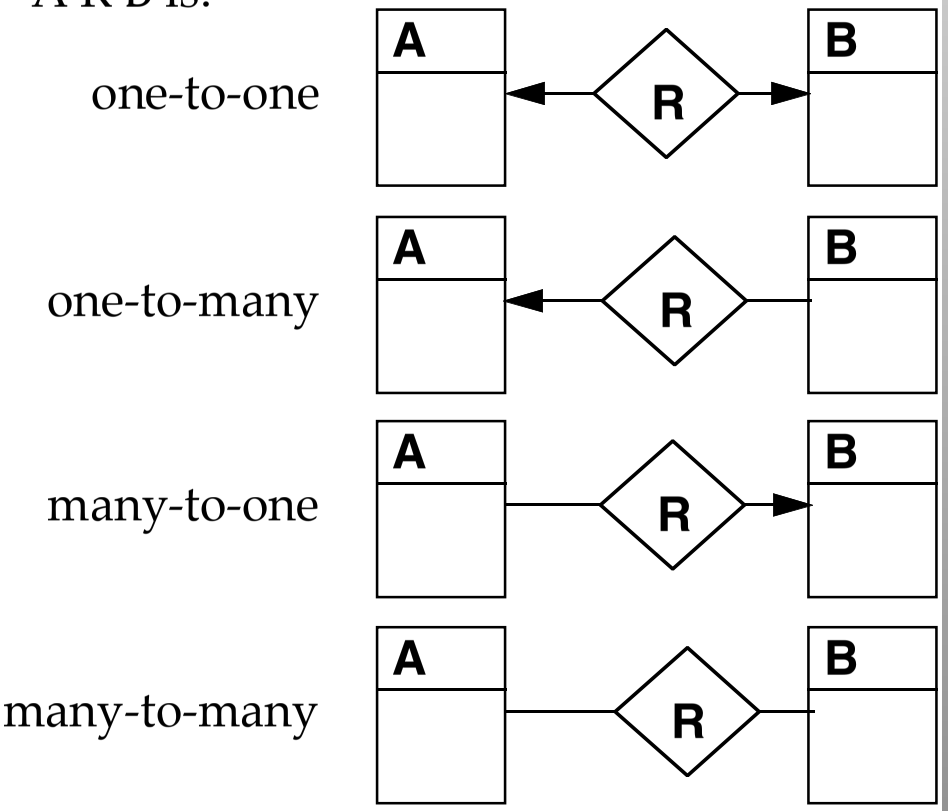
Entity & Attributes



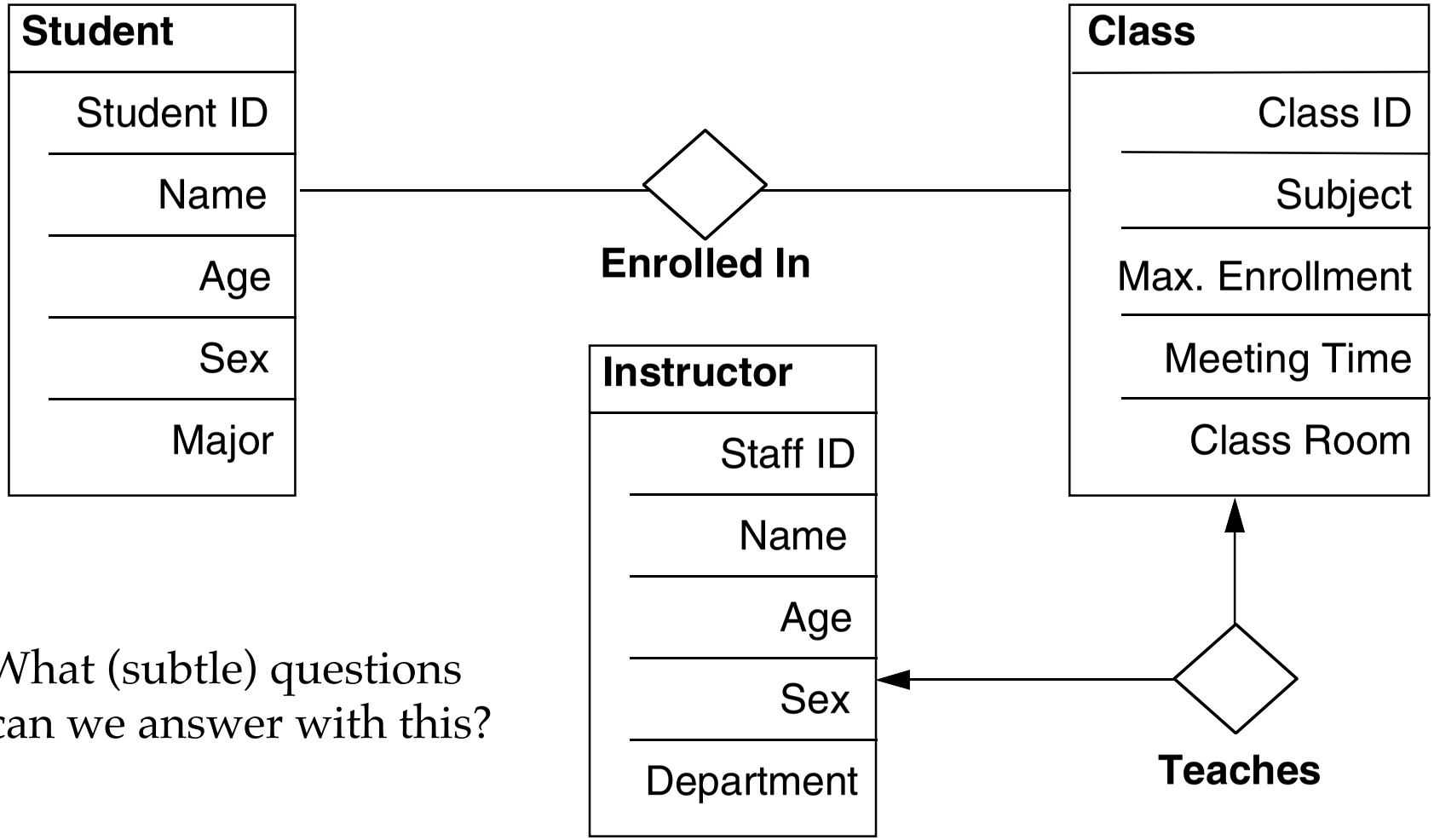
A bit different from the notation used in [GJM].

Relationship

A R B is:



ER Diagram Example



What (subtle) questions can we answer with this?

Characteristics of ER Models

- Good to answer questions on the relationships among entities.
  - How do we get from here to there?
  - If we add/delete this, what else do we need to do?
- Not well standardized.
  - E.g. some allow n-ary relations, and/or inheritance through an is-a relation.
- Constraints on relationship are hard to express.
  - They often need to be expressed separately.
  - E.g. [GJM] Exercise 5.26.

Logic Specifications

- Express conditions and constraints satisfied using first-order predicate logic.
  - $\forall, \exists$
  - $\&, |, \neg, \supset, \equiv$
  - Predicates.
- Example:
  - $\forall c \in$ **Class**.
    $5 \leq |\{s \mid \langle s, c \rangle \in$ **Enrolled_In**$\}| \leq$a.Max_Enrollment
- Assertions
  - State pre- and post-conditions of a program segment.
  - {Pre} P {Post}
  - Exercise 5.32, page 207 [GJM]

Algebraic Specifications

- "Larch-like" notation described in [GJM]
  - Define software systems as heterogeneous algebra, i.e., "collection of sets together with operations."

```
algebra StringSpec;
introduces
   sorts String, Char, Nat, Bool;
   operations
      new: () → String;
      append: String, String → String
      add: String, Char → String;
      length: String → Nat;
      isEmpty: String → Bool;
      equal: String, String → Bool;
constrains new, append, add, length
   isEmpty, equal so that
for all [s, s1, s2: String, c: Char]
   isEmpty (new ()) = true;
   isEmpty (add (s, c)) = false;
   length (new ()) = 0;
   length (add (s, c)) = length (s) + 1;
   append(s, new ()) = s;
   append(s1, add(s2,c)) =
      add(append(s1, s2), c);
   equal (new(), new()) = true;
   equal (new(), add(s, c)) = false;
   equal (add(s,c), new ()) = false;
   equal (add(s1, c), add(s2, c)) =
      equal(s1, s2);
end StringSpec
```