Overview of Chapter 5

- What is a Specification?
- Operational and Diagrammatic Specifications
  - Data Flow Diagrams, Finite State Machines, Petri Nets, Entity Relationship Diagrams
  - UML Diagrams (Briefly – Revisit in Total)
- Mathematical-Based Specifications
  - Queuing and Simulation Models
  - Declarative Specifications
    - Logic-Based Notations
    - Algebraic Notations
  - Languages for Modular Specifications
    - Statecharts and Z
- Specification Notations and Writing Specifications
What’s in a Specification?

- Specification is a Broad Term that Means Definition
  - Used at Different Stages of Software Development For Different Purposes
  - Statement of Agreement (Contract) Between
    - Producer and Consumer of a Service
    - Implementer and User
  - All Desirable Qualities Must Be Specified
- Statement of User Requirements
  - Major Failures Occur Due to Misunderstandings Between Producer and User
  - "The Hardest Single Part of Building a Software System is Deciding Precisely What To Build" (F. Brooks – Mythical Man Month)

What’s in a Specification?

- Specification can be Many Different things to Many Different Stakeholders (User, Designer, Manager, etc.)
  - Requirements Specification: Agreement between End User and Designers
  - Design Specification: Agreement between Designers and Developers
  - Module Specification: Agreement between SEs that Use a Module and SEs that Implement a Module re. Interface
  - Object-Oriented Specification: Assertion of Class’ Capability via Public Interface
- Specification Involves “What”
- Implementation Involves “How”
Uses of Specification

- Statement of User’s Needs
  - Statement of Interface Between Machine and Controlled Environment
- Critical Issues:
  - User’s Needs May not be Understood by Developer
  - Need Ability to Verify Specification
- Problem Faced:
  - Serious Undesirable Effects can Result Due to Misunderstandings Between Software Engineers and Domain Experts

Uses of Specification

- Statement of Implementation Requirements
  - Hardware, OS, Language(s), DBMS, etc.
  - Requirement Specification
    - External Behavior of System
    - Functional and Non-Function Behavior
    - Design Spec Verified Against Requirements Spec
  - Design Specification
    - Description of Software Architecture
    - Code Must be Verified Against Design Spec
- Reference Point During Product Maintenance
  - Verify that Changes Satisfy Requirements
  - Change in Specification Requires Adaptive Maintenance
Classic Information System Design

Data vs. Information

- ANSI Definitions
  - Data
    - A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means.
    - Any representation such as characters or analog quantities to which meaning is or might be assigned. Generally, we perform operations on data or data items to supply some information about an entity.
  - Information
    - Meaning that a human assigns to data by means of the known conventions used in their representation.
Specification Qualities

- Clear, Unambiguous, Understandable
- Consistent
  - Inconsistencies May be Impossible to Implement
  - Complexity May Lead to Inconsistency
  - Inconsistencies Lead to Incorrect Implementation
- Completeness
  - Internally Complete: Specification Defines any new Concept or Terminology that it Uses
  - W.r.t. Requirements; All Requirements Should be Contained within Specification
  - Incrementality Aids Achievement of Completeness
- How Does the Achievement of Software Qualities Affect the Attainment of Specification Qualities?
- Let’s See Some Examples of These Qualities…

Clear, Unambiguous, Understandable

- Specification Fragment for a Word-Processor
- Can an Area be Scattered, or Must it be Contiguous?

Selecting is the process of designating areas of the document that you want to work on. Most editing and formatting actions require two steps: first you select what you want to work on, such as text or graphics; then you initiate the appropriate action.
Precise, Unambiguous, Clear

- Consider a Real-Time Safety-Critical System
  - Can a message be accepted as soon as we receive 2 out of 3 identical copies of message or do we need to wait for receipt of the 3rd?

  The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.

Consistent

- Specification Fragment for a Word-Processor
  - What if the length of a word exceeds the length of the line?
  - What should the action of the editor be?

  The whole text should be kept in lines of equal length. The length is specified by the user. Unless the user gives an explicit hyphenation command, a carriage return should occur only at the end of a word.
Specification Styles

- Informal: Natural Language, Spec by Visio/PPT, Figures, Tables, etc.
- Formal: Notation with precise Syntax/Semantics
  - Advantages:
    - May Allow Formal Verification
    - May Support Automatic Processing (Code Gen)
    - Allows Use of Mathematical Models
    - May be Used to Generate Test Cases (Chapter 6)
  - Disadvantages:
    - Formal Specifications Not Widely Used
    - Hard to Justify Economic Gain
    - Training Issues for Personnel
- Semi-Formal: No Precise Semantics (TDN/GDN)

Specification Styles

- Operational
  - Describes Desired Behavior of System
  - Usually Provides a Model of System Behavior
  - Verified by Prototyping
- Descriptive
  - Describes Desired Properties of System
  - Declarative Specification
  - Usually Uses Mathematical Equations
  - More Abstract than Operation Specification
  - No Focus on Implementation
- Actual Specifications – Mix of Both…
Operational Specification Example

Consider a geometric figure E:

E can be drawn as follows:
1. Select two points $P_1$ and $P_2$ on a plane
2. Get a string of a certain length and fix its ends to $P_1$ and $P_2$
3. Position a pencil as shown in next figure
4. Move the pen clockwise, keeping the string tightly stretched, until you reach the point where you started drawing

Verification of Specifications

Two Approaches:
- “Observe” Dynamic Behavior of Specified System (Simulation, Prototyping, “Testing” specs)
- Analyze Properties of the Specified System

Both Depend on Formality of Specification

Analogy with Traditional Engineering
- Physical Model of a Bridge
  - Build An Actual Model
  - Test it out in Wind Tunnel
- Mathematical Model of a Bridge

What’s Reality of Bridge Building?
- Guarantee of Success w.r.t. Weight, Weather, etc.

Are there Similar Guarantees in Software?

What Software Applications Need Such Guarantees?
Operational Specifications

- Allow the Behavior of a System to be Defined
- Many Complementary Techniques
  - Data Flow Diagrams
  - UML Diagrams (Briefly and Separate Lecture)
  - Finite State Machines
  - Petri Nets
- Operational Specifications Provide Means to Model System from Alternative Perspectives
  - Perspectives Must be Consistent with One Another
  - Some Techniques Target End-User/Customer
  - Others are More Software Engineering Intensive
  - Key Issue: All Diagrams Must be Consistent in Terminology to Yield Strong Result

Data Flow Diagrams (DFDs)

- A Semi-Formal Operational Specification
- System Viewed as Collection of Data Manipulated by “Functions”
- Data can be Persistent - Stored in Data Repositories
- Input State to Represent Trigger of Data Flow
- Output State(s) to Represent Result of Data Flow
- Data can Flow from Input to Function to/from Repositories to Output
- DFDs have a Graphical Notation
  - Tools are Commercially Available
**DFDs Employ a Graphical Notation**

- What are Main Modeling Constructs?
  - Bubbles Represent Functions
  - Arcs (Arrows) Data Flows
  - Open Boxes Represent Persistent Store
  - Closed Boxes Represent I/O Interaction

- Note: DFDs can not Represent Sequential Steps

---

**Methodology of Information System**

1. Start from the “context” diagram

   ![Diagram of information system](image)
Methodology of Information System

2. Proceed by refinements until you reach “elementary” functions ( preserve balancing)

A Library Example DFD

- Book
- Title
- Author
- List of Authors
- List of titles
- List of topics

Get a book

Book request by the user

Book title; user name

List of books borrowed

Display of the list of titles
Refinement of “Get a book”

Patient Monitoring Systems

The purpose is to monitor the patients’ vital factors--blood, pressure, temperature, ...--reading them at specified frequencies from analog devices and storing readings in a DB. If readings fall outside the range specified for patient or device fails an alarm must be sent to a nurse. The system also provides reports.
A Refinement as Detailed DFD

Further Refinement
A Lower-Level DFD for HTSS

Evaluating DFDs

- Easy to Read, but ...
- Informal Semantics
  - How Does One Define Leaf Functions?
  - Inherent Ambiguities in Flow
- Consider the DFD (Functions) Below:

- Are Outputs from A, B, C all needed Before D is Enabled?
- What if Only A and B Present?
- Do A, B, and C have to be Received in a Particular Order?
- Outputs for E and F are produced at the same time?
Evaluating DFDs

- Clearly, Control Information is Absent

\[ A \rightarrow B \]

Possible interpretations:
(a) A produces datum, waits until B consumes it
(b) B can read the datum many times without consuming it
(c) a pipe is inserted between A and B

- DFDs are a Semi-Formal Notation
  - DFDs by Visio and PPT
  - Excellent Vehicle for Presenting to End-User
  - Not Standalone – Need Complementary Diagram(s) to Fully Specify System Capabilities

Finite state machines (FSMs)

- Utilized to Specify control flow aspects
- An FSM Consists of:
  - A finite set of states (nodes), Q;
  - A finite set of inputs (labels), I;
  - A transition function \( d : Q \times I \rightarrow Q \) (\( d \) can be a partial function)
- FSMs are Well Suited to Represent Systems with
  - Multiple Known States
  - Well-Defined Events for State Changes
Classic FSM Examples

On
Push switch
Off

High-pressure alarm

On
Push switch
Off

High-temperature alarm

Restart

FSM’s Can Model Real World

Consider a Refinement of High Pressure/High Temperature FSM on Previous Slide

Pressure signal

Pressure action

Temperature signal

Normal

Successful recovery

Successful recovery

Temperature signal

Temperature action

Off

Unsuccessful recovery

Unsuccessful recovery

Pressure signal
FSM for HTSS

- Totals a Customer’s Order

Classes of FSMs

- Deterministic/Nondeterministic
- FSMs as Recognizers - Introduce Final States
- FSMs
  - Used for Lexical Analysis in Compilers
  - Realize Regular Expressions in Automata

$q_0$ is an initial state
$q_f$ is a final state
FSMs as Recognizers

FSMs as transducers - introduce set of outputs

Legend:
- `<letter>` is an abbreviation for a set of arrows labeled a, b, ..., z, A, ..., Z.
- `<digit>` is an abbreviation for a set of arrows labeled 0, 1, ..., 9, respectively.

FSMs Usage and Limitation

- FSMs are Simple and Widely Used
  - Control Systems, Compilation
  - Pattern Matching, Hardware Design
- Most Software Applications can be Modeled via FSMs
- In Practice, FSMs are Good for Sample or Portions of System – Problems Can Arise:
  - Model Different Portions of Application
  - Compose FSMs for Entire System View
  - Finite Memory Yields State Explosion
    - Suppose n FSMs with $k_1$, $k_2$, ..., $k_n$ states
    - Composition is a FSM with $k_1 \times k_2 \times \ldots \times k_n$. This growth is exponential with the number of FSMs, not linear (we would like it to be $k_1 + k_2 + \ldots + k_n$)
Example of State Explosion:

Consider Three Individual FSMs:

Producer

Consumer

Composition: The Resulting FSM

Clearly, Complexity Has Increased
Petri Nets

- Petri Nets are another graphical formalism for system’s specification consisting of:
  - Finite set of places (circles – Pᵢ’s)
  - Finite set of transitions (horizontal lines – tⱼ’s)
  - Finite set of arrows connecting places to transitions and transitions to places
  - Tokens (dots)

Petri Nets – Other Concepts

- Marking: Assigning non-negative integers to places which represent tokens in the network
- State: Petri Net with marking
- Enabled: Transition if all of its incoming places have at least one token
- Fire: Transition consumes token from incoming places and produces token for outgoing places
- Firing sequence: Sequence of transition firings for execution of PN (t₁, t₃, t₂, t₅, ...)
- PNs are non-deterministic
  - Any enabled transition may fire
  - Neither when nor which fires is specified by model
**Modeling with Petri nets**

- Places Represent Distributed States
- Transitions Represent Actions Or Events That May Occur When System is in a Certain State
- They Can Occur as Certain Conditions Hold on the States
- Forks and Joins Modeled
  - Fork: Transition from 1 Input to N Outputs
  - Join: Transition from N Inputs to 1 Output

---

**Petri Nets**

- A Petri Net is Defined as a Quadruple \((P,T,F,W)\) with:
  - \(P\): places \(T\): transitions (\(P, T\) are finite)
  - \(F\): flow relation (\(F \subseteq \{P \times T\} \cup \{T \times P\}\) )
  - \(W\): weight function (\(W: F \rightarrow N - \{0\}\) )
- Properties:
  1. \(P \cap T = \emptyset\)
  2. \(P \cup T \neq \emptyset\)
  3. \(F \subseteq (P \times T) \cup (T \times P)\)
  4. \(W: F \rightarrow N - \{0\}\)
- Default Value of Weight Function \(W\) is 1
- State Defined by Marking: \(M: P \rightarrow N\)
PN Semantics: Dynamic Evolution

- Transition $t$ is Enabled iff
  - $\forall p \in t$'s Input Places, $M(p) \geq W(<p,t>)$

- Transition $t$ Fires: Produces a New Marking $M'$ in Places that are Either $t$'s Input or Output or Both
  - If $p$ is an Input Place: $M'(p) = M(p) - W(<p,t>)$
  - If $p$ is an Output Place: $M'(p) = M(p) + W(<t,p>)$
  - If $p$ is both an Input and an Output Place:
    - $M'(p) = M(p) - W(<p,t>) + W(<t,p>)$

Graphical Representation of PN

- Place $P_1$
- Place $P_2$
- Place $P_3$
- Place $P_4$
- Place $P_5$
- Place $P_6$
- Place $P_7$
- Transition $t_1$
- Transition $t_2$
- Transition $t_3$
- Transition $t_4$
- Transition $t_5$
- Transition $t_6$
After (a) either (b) or (c) may occur, then (d)

Modeling Concurrent Systems

- **Concurrency**: Two Transitions are Enabled to Fire in Given State, and the Firing of One Does Not Prevent the Other From Firing
  - See $t_1$ and $t_2$ in Case (a)

- **Conflict**: Two Transitions are Enabled to Fire in Given State, But the Firing of One Prevents the Other From Firing
  - See $T_3$ and $T_4$ in Case (d)
  - Place $P_3$ Models Shared Resource Between Two Processes

- No Policy Exists to Resolve Conflicts (Known as Unfair Scheduling)

- A Process May Never Get a Resource (Starvation)

- Deadlock: A Marking Where no Transition May be Enabled
Avoiding Starvation

- Focus on Cycle in Middle of PN

A Conflict-Free PN

- Always a Token Available in Place R for Both Sides to Utilize - R Reloaded with 2 Tokens at Each Step

This net can deadlock! Consider \( \langle t_1, t_3, t_2, t_4 \rangle \)
A Deadlock-Free PN

- If One Side Uses 2, it Proceeds, Else Other Side
- Starvation is Still Possible

\[ \text{A Partial Starvation} \]
- Place Supplying t4 is Not Refilled with Token
Producer-Consumer Example

- Individual PNs for Produce and Consume

Separate nets for the subsystems

- Combined PNs

One net for the entire system
Petri Net Limitations

- Petri Nets are Limited in Their Modeling Capability
  - As Presented, Non-Deterministic
  - No Way to Prioritize Among All Eligible Active Transitions that Can Fire
- There are a Number of Capabilities that Would be Very Useful in Modeling System Behavior
  - Adding Predicates and Functions that are Used to Evaluate Conditions Under Which a Transition Fire and its Results
  - Instituting Priority to Decide When to Fire Among Multiple Transitions that are Eligible
  - Incorporating Timing to Constrain When a Transition can Fire

Assigning Values to Tokens

- Transitions have Predicates and Functions
- Predicate Refers to Values of Tokens in Inputs
- Functions Define Values of Tokens for Outputs

The firing of \( t_1 \) by using \(<3,7>\) would produce the value 10 in \( P_4 \). \( t_2 \) can then fire using \(<4,4>\)
Other PN Extensions

Specifying Priorities
- Function Pri from Transitions to Natural Numbers:
  \[ \text{pri}: T \rightarrow N \]
- When Several Transitions are Enabled, Only Ones with Maximum Priority are Allowed to Fire
- If Multiple Active, choose Non-deterministically

Timing
- Pair of Constants \(<t_{\text{min}}, t_{\text{max}})>\) associated with each Transition – If Transition Enabled
  - Must wait for at least \(t_{\text{min}}\) to elapse before it can Fire
  - Must Fire before \(t_{\text{max}}\) has elapsed, unless it is Disabled by the Firing of another Transition before \(t_{\text{max}}\)

Combining Timing and Priorities

\[ \begin{align*}
  p_1 &: t_1, t_{\text{min}} = 1, t_{\text{max}} = 4, \text{priority} = 1 \\
  p_2 &: t_2, t_{\text{min}} = 2, t_{\text{max}} = 3, \text{priority} = 3 \\
  p_3 &: t_3, t_{\text{min}} = 0, t_{\text{max}} = 5, \text{priority} = 2
\]
Overview of a PN Example

- PNs can be Used for Very Complex Applications
- Consider
  - An N Elevator System to be Installed in a Building with M Floors
  - Natural Language Specifications Contain Several Ambiguities
  - Formal Specification Using PNs Removes Ambiguities
- How is a Solution Constructed?
  - Employ Modules
  - Each Encapsulating Fragments of PNs
  - Each Captures Certain System Components

Initial Sketch of Movement

- Elevator at floor \( j \)
- Elevator at floor \( j + 1 \)
- Transfer from floor \( j \) to \( j + 1 \)
- Pushing internal button for floor \( j + 1 \)
- Button illumination
Dealing with Buttons

**Internal**

- Fj
- ILBj
- Set
- 0.0
- Reset
- On
- Off

**External**

- Fj'
- UPj
- t1
- On
- Off
- x.x Reset
- t2

Modeling the Entire PN
Entity Relationship Diagrams

- **ER Model Concepts**
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types

- Relationships of Higher Degree

- Skip Extended Entity-Relationship (EER) Model

- Notation is based on:

Summary of ER-Diagram Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENTITY TYPE</td>
</tr>
<tr>
<td></td>
<td>WEAK ENTITY TYPE</td>
</tr>
<tr>
<td></td>
<td>RELATIONSHIP TYPE</td>
</tr>
<tr>
<td></td>
<td>IDENTIFYING RELATIONSHIP TYPE</td>
</tr>
<tr>
<td></td>
<td>ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>KEY ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>MULTIVALUED ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>COMPOSITE ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>DERIVED ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>TOTAL PARTICIPATION OF E₁ IN R</td>
</tr>
<tr>
<td></td>
<td>CARDINALITY RATIO 1:N FOR E₁,E₂ IN R</td>
</tr>
<tr>
<td></td>
<td>STRUCTURAL CONSTRAINT (min, max) ON PARTICIPITION OF E IN R</td>
</tr>
</tbody>
</table>
Requirements of the Company (Oversimplified for Illustrative Purposes)

- Company is Organized into Departments
  - Each Department has a Name, Number and an Employee Who Manages the Department
  - We Track of the Start Date of the Department Manager

- Each Department Controls a Number of Projects
  - Each Project has a Name, Number and is Located at a Single Location

Store Each Employee’s Social Security Number, Address, Salary, Sex, and Birthdate

- Each Employee Works for One Department but May Work on Several Projects
- We Track of the Number of Hours Per Week that an Employee Currently Works on Each Project
- We Track of the Direct Supervisor of Each Employee

- Each Employee May have a Number of Dependents
  - For Each Dependent, We Track of their Name, Sex, Birthdate, and Relationship to Employee
ER Model Concepts: Entities and Attributes

- Entities - Specific Objects or Things in the Mini-world that are Represented in the Database
  - EMPLOYEE John Smith
  - Research DEPARTMENT
  - Productx PROJECT

- Attributes are Properties Used to Describe an Entity.
  e.g., an EMPLOYEE Entity may have a Name, SSN, Address, Sex, Birthdate

- A Specific Entity (Instance) has a Value for Each of its Attributes
  - Specific Employee Entity May Have Name=’John Smith’, SSN=’123456789’, Address=’731 Fondren, Houston, TX’, Sex=’m’, Birthdate=’09-jan-55’
Three Types of Attributes

- Simple: Single Atomic Value for the Attribute
  - SSN or Sex or State or Salary or ...
- Composite: Attribute Composed of Many Components
  - Address (Apt#, House#, Street, City, State, Zipcode, Country) or Name(Fname, MI, Lname)
  - Composition May form a Hierarchy where Some Components are Themselves Composite
- Multi-Valued: Entity may have Multiple Values for That Attribute - Like an Set Type
  - CAR {Color} or STUDENT {Previousdegrees}

Composite and Multi-valued Attributes may be Nested Arbitrarily to any Number of Levels (Rare)
- Previousdegrees of a STUDENT is a Composite Multi-valued Attribute Denoted by
  {Previousdegrees(college, Year, Degree, Field)}

Entities with Attribute Values

- Name = John Smith
- Address = 2311 Kirby, Houston, Texas 77001
- Age = 55
- HomePhone = 713-749-2630
- Name = Sunoco Oil
- Headquarters = Houston
- President = John Smith
Entity Types and Key Attributes

- Entities with the Same Basic Attributes Are Grouped or Typed into an **Entity Type**
  - EMPLOYEE Entity Type or PROJECT Type
- Attribute of Entity Type for which Each Entity Must Have a Unique Value is Called a **Key Attribute**
  - SSN of EMPLOYEE, ISBN of BOOK
- A Key Attribute may be Composite
  - VIN is a Key of the CAR Entity Type
- An Entity Type may have More than One Key
  - CAR Entity Type May Have Two Keys:
    - VIN
    - Vehicletagnumber (Number, State) aka License Plate

Entity Type CAR with Attributes

- **CAR**
- Registration(RegistrationNumber, State), V_ID, Make, Model, Year, (Color)

- \( car_1 \) ((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1989, (red, black))
- \( car_2 \) ((ABC 123, NEW YORK), WP9872, Nissan Sentra, 2-door, 1992, (blue))
- \( car_3 \) ((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 1993, (white, blue))
Two Other Entity Types

<table>
<thead>
<tr>
<th>ENTITY TYPE NAME:</th>
<th>EMPLOYEE</th>
<th>COMPANY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name, Age, Salary</td>
<td>Name, Headquarters, President</td>
</tr>
<tr>
<td>ENTITY SET:</td>
<td>( e_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>(EXTENSION)</td>
<td>( (\text{John Smith, 55, 80k}) )</td>
<td>( (\text{Sunco Oil, Houston, John Smith}) )</td>
</tr>
<tr>
<td></td>
<td>( e_2 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td></td>
<td>( (\text{Fred Brown, 40, 30k}) )</td>
<td>(Fast Computer, Dallas, Bob King)</td>
</tr>
<tr>
<td></td>
<td>( e_3 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( (\text{Judy Clark, 25, 20k}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>

Relationships and Relationship Types

- A Relationship Relates Two or More Distinct Entities With a Specific Meaning
  - \text{EMPLOYEE John Smith Works on the Productx PROJECT}
  - \text{EMPLOYEE Franklin Wong Manages the Research DEPARTMENT}
  - Relationship - Instance Level
- Relationships of the Same Type are Grouped or Typed Into a Relationship Type
  - \text{WORKS\textunderscore ON Relationship Type in Which Employees and Projects Participate}
  - \text{MANAGES Relationship Type in Which Employees and Departments Participate}
  - Analogous to Reference or List in Programming
The WORKS_ON Relationship

Employee | Works On | Project
---|---|---
E1 J. Doe | | P1 Instrumentation
E2 M. Smith | | P2 Database Develop.
E3 A. Lee | | P3 CAD/CAM
E4 J. Miller | | P4 Maintenance
E5 B. Casey | | 
E6 L. Chu | | 
E7 R. Davis | | 
E8 J. Jones | | 

Relationships and Relationship Types

- Degree of a Relationship Type is the Number of Participating Entity Types
  - Both MANAGES and WORKS_ON are Binary Relationships
  - What is a possible Ternary Relationship?
- More Than One Relationship Type Can Exist With the Same Participating Entity Types
  - MANAGES and WORKS_FOR are Distinct Relationships Between EMPLOYEE and DEPARTMENT Entity Types
- Relationships are Directional
  - SUPPLIES: SUPPLIER to PARTS
  - SUPPLIERS: PARTS to SUPPLIER
Weak Entity Types

- Entity that Does Not have a Key Attribute
- Weak Entity Must Participate in an Identifying Relationship Type with an Owner or Identifying Entity Type
- Entities are Identified by the Combination of:
  - A Partial Key of the Weak Entity Type
  - Particular Entity they Are Related to in the Identifying Entity Type
- Example:
  - A DEPENDENT Entity is Identified by Dependent’s First Name and Birthdate, and the EMPLOYEE That the Dependent is Related to
  - DEPENDENT is a Weak Entity Type With EMPLOYEE as its Identifying Entity Type Via the Identifying Relationship Type DEPENDENT_OF
ER Model and Data Abstraction

- Abstraction
- Classification
- Aggregation
- Identification
- Generalization

ER Model Concept
- Entity Type - a Grouping of Member Entities
- Relationship Type - a Grouping of Member Relationships
- Relationship Type is an Aggregation of (Over) Its Participating Entity Types
- Weak Entity Type and Attribute Key

Constraints on Aggregation

- Cardinality Constraints on Relationship Types
  - AKA Ratio Constraints
  - Maximum Cardinality
    - One-to-One
    - One-to-Many
    - Many-to-Many
  - Minimum Cardinality (AKA Participation or Existence Dependency Constraints)
    - Zero (Optional Participation, Not Existence-Dependent)
    - One or More (Mandatory, Existence-Dependent)
One-to-many (1:N) or Many-to-one (N:1)

Many-to-Many (M:N)
One-to-One Relationship

- Each Instance of One Entity Class E1 Can Be Associated with Exactly One Instance of Another Entity Class E2 and Vice Versa.
- Example:
  - Each Employee Can Work in Exactly One Project and Each Project Employs Exactly One Employee

One-to-One WORKS_ON Relationship

- WORKS_ON Relationship Instances
- EMPLOYEE Set
- PROJECT Set
**Many-to-One Relationship**

- Each Instance of One Entity Class E1 can be Associated with Zero or More Instances of Another Entity Class E2, but Each Instance of E2 can be Associated With at Most 1 Instance of E1
- Example:
  - Each Employee Can Work in Exactly One Project
  - Each Project Can Employ Many Engineers

**One-to-Many WORKS_ON Relationship**

- EMPLOYEE Set
- PROJECT Set
- WORKS_ON Relationship Instances
Many-to-Many Relationship

- Each Instance of One Entity Class Can Be Associated with Many Instances of Another Entity Class, and vice versa
- Example:
  - Each Employee Can Work in Many Projects
  - Each Project Can Employ Many Employees

Many-to-Many WORKS_ON Relationship

- Relationship Instances
- EMPLOYEE Set
- PROJECT Set
Structural Constraints

- Structural Constraints on a Relationship are One Way to Express the Semantics of a Relationship
- Cardinality Ratio (of a Binary Relationship): 1:1, 1:N, N:1, or M:N
  - Shown by Placing Apropos Number on the Link
- Participation Constraint (on Each Entity Type):
  - Total (Called Existence Dependency) or Partial
  - Shown By Double Lining The Link
- NOTE:
  - Easy to Specify for Binary Relationship Types
  - Do Not Be Misled by Obscure Notations to Specify Above Constraints for Higher Order Relationships

Relationships of Higher Degree

- Relationship Types of Degree 2 Are Called Binary
- Relationship Types of Degree 3 Are Called Ternary
  - There is a Concrete Relationship Instance that Involves all Three Entity Types
  - These are Not Separate Relationships!
- Relationship Types of Degree N Are Called N-ary
  - Again - Concrete n-Participation Relationship
- In General, an N-ary Relationship is Not Equivalent to N Binary Relationships
  - Rather - it is more Analogous to the Grouping of N-Binary Relationships into a N-ary Relationship
Ternary Relationships

![Diagram of ternary relationships between SUPPLIER, SUPPLY, PROJECT, PART, CAN_SUPPLY, USES, and PRO_NAME.]

Ternary Relationships

![Diagram of ternary relationships between SUPPLIER, SUPPLY, PROJECT, PART, CAN_SUPPLY, USES, and PRO_NAME.]

CHS 51

CHS 52
Ternary vs. Binary Relationships

Ternary Relationships

Ternary Relationships - Instances

SUPPLIER

SUPPLY

PROJECT

PART

s1
s2

r1
r2
r3
r4
r5
r6
r7

j1
j2
j3

p1
p2
p3

CSE230
Modified Earlier Example

Another ER Diagram - Bank Example
What are Problems with ER Notation?

- Historically, ER Model 1st Proposed in 1976
- However, ER Model in this Original Form Did Not Support the Generalization Abstraction (Inheritance)
- In Databases, Inheritance 1st Proposed in 1977
- Thus, Extended ER Evolved through 1980s with the Focus on “Semantic Data Models”

Specialization/Attribute Inheritance

- An Entity Type E1 is a **Specialization** of another Entity Type E2 if E1 has the Same Properties of E2 and Perhaps Even More.
- E1 IS-A E2
Generalization

Abstracting the Common Properties of Two or More Entities to Produce a “Higher-level” Entity

<table>
<thead>
<tr>
<th>Project Office</th>
<th>Specialty Office</th>
<th>Region Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineer</td>
<td>Secretary</td>
<td>Salesperson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee No</th>
<th>Employee Name</th>
<th>Salary</th>
<th>Title</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Employee No</td>
<td>Employee Name</td>
<td>Salary</td>
</tr>
<tr>
<td>Employee No</td>
<td>Employee Name</td>
<td>Salary</td>
<td>Title</td>
<td>Address</td>
</tr>
<tr>
<td>Employee No</td>
<td>Employee Name</td>
<td>Salary</td>
<td>Title</td>
<td>Address</td>
</tr>
</tbody>
</table>

EMPLOYEE

Generalization

EMPLOYEE

<table>
<thead>
<tr>
<th>Employee No</th>
<th>Employee Name</th>
<th>Salary</th>
<th>Title</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineer</td>
<td>Secretary</td>
<td>Salesperson</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Project Office | Office | Specialty | Office | Region | Car |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineer</td>
<td>Secret</td>
<td>Salesperson</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CSE230
Constraints

- disjoint, total
- disjoint, partial
- overlapping, total
- overlapping, partial

Part

\[
\text{Manufactured Part} \quad \text{Purchased Part}
\]

\[
\text{PartNo} \quad \text{Description} \quad \text{SupplierName} \quad \text{ListPrice}
\]

\[
\text{Part} \cup \text{Manufactured Part} \cup \text{Purchased Part}
\]

Total and Partial Disjoint

Employee

\[
\text{Hourly Rate} \quad \text{Employee No} \quad \text{Employee Name} \quad \text{Salary}
\]

\[
\text{Title} \quad \text{Address}
\]

\[
\text{HOURLY_EMP} \quad \text{SALARIED_EMP}
\]

\[
\text{Salary}
\]

\[
\text{Engineer} \quad \text{Secretary} \quad \text{Salesperson}
\]

\[
\text{Project} \quad \text{Office} \quad \text{Specialty} \quad \text{Office} \quad \text{Region} \quad \text{Car}
\]
Total Overlapping

PTT

MANUFACTURED_PART

Purchased_PART

Batch No. Drawing No.

Price

HTSS ER Diagram Example

Customer Order

Order

Deli Order

Item

Sales

Daily Sales

Total Date

ISA

Sales

Item

Deli Item

Weight

Cost

Cost Inc.

Increment

Total

Customer

AcctNum

Total

UPC

WCost

RCost

Size

OnStock

InStock

Location

RULimit

Name

HTSS ER Diagram Example

Interplay of Specification Techniques

- What do DFDs have to Offer re. OO Design?
  - Data Stores (Classes)
  - Arrow Labels (Parameters)
  - Input (Method of Class)
  - Functions (Implementation of Method of Class)
  - Output (Return of Method of Class)
- What do DFDs have to Offer re. ER Design?
  - Data Stores (Entities)
  - Arrow Labels (Relationships and Keys)
- What about FSMs?
- What is Impact w.r.t. Modular or ADT/Class Design?
Interplay of Specification Techniques

CSE230

Interplay of Specification Techniques

CH5.108

CSE230

Interplay of Specification Techniques

CH5.107
UML Diagrams

- UML is a Language for Specifying, Visualizing, Constructing, and Documenting Software Artifacts
- UML Formalizes the Previous Techniques (DFD, ER, FSM, PN, etc.) into a Unified Environment
- What Does a Modeling Language Provide?
  - Model Elements: Concepts and Semantics
  - Notation: Visual Rendering of Model Elements
  - Guidelines: Hints and Suggestions for Using Elements in Notation
- UML has 13 Different Diagrams (2.0)

References and Resources
- Web for UML 2.0: www.uml.org

UML Use-Case Diagrams

- Define Functions on Basis of Actors and Actions

[Diagram showing use-case scenarios for a library system]
UML Sequence Diagrams

- Describe Object Interactions by Exchanging Messages

**Librarian Catalogue**

- member card + book request
  - membership OK
  - book request
  - book available
  - book borrowed

**Customer**

**Catalogue**
### UML Collaboration Diagrams

**Object Interactions and Their Ordering**

1. member card + book request → Customer
2. membership OK
3. book request → Librarian
4. book available → Catalogue
5. book borrowed

### UML Statechart Diagrams

**Akin to Finite State Machine**
Activity Diagram

- Akin to Petri Net

- Let's Consider UML in Total (Jump to New PowerPoint Presentation)

Mathematical-Based Specifications

- Queueing and Simulation Models
  - Predict and Simulate System Behavior
  - CSE221

- Declarative Specifications:
  - Logic Specifications
  - Algebraic Specifications
  - CSE233 Programming Languages

- Languages for Modular Specifications
  - Statecharts and Z