Requirements Cont.

- Introduction: Synchronization
  - where are we now
- Data-flow Diagrams
- Entity-Relationship Diagrams

Synchronization

- Note text chapters and other assigned readings
  - keep up with readings and journals
  - check the syllabus on line for further information
- Will assign each manager (or designate) a presentation at some point soon.
  - **status report** to class
  - **evaluation** of strengths/weaknesses of group structure
    - and management style
  - **recommendation** for improvement
    - class input to be recorded

Next week

- Groups to produce a Requirements document from the template
  - carefully track TBD's
    - ¤ assign responsibilities and schedule resolutions
  - refine the V & S and the Risk Document as you go
  - focus on Use Cases for User Requirements
    - ¤ moves us towards Functional requirements
  - I suggest finding good examples of Requirements documents
    - ¤ besides those on my site
- Continue to **elicit** requirements (use cases and DD)
  - customer (and other stakeholder) contact

Suggestions (for good / passing grades)

- Subordinate "student" vs. "professional" attitude
  - fulfillment of a goal you define
  - a "minimal" satisfaction of stated guidelines is not good
  - passing grades mean ability to add value to the team effort through processes we discuss in class
    - as has been said, easy to fail by flaking on the team, leading the team astray
    - crashing and burning project not necessarily failure at all
    - team individual evaluations next week
- Check my 308 page for standards
  - pay close attention to each item, discuss relevance
  - look all around, follow links, figure out what is there!

Specific pointers

- Date ALL documents
- Author ALL documents
- Assign responsibilities and schedule followups
  - enforce responsibility
  - can be lenient and trade off favors, but not one-sided!
  - journals contain action items and meeting notes
- Re-read and **implement** strategies discussed in class and in the book
  - how many groups have used the customer bill of rights and responsibilities?
    - ¤ is your customer a bit flakey?
    - ¤ can you work with that to YOUR advantage?

More

- Prioritize **everything** (like scheduling!)
  - internal jobs, requirements, etc.
- Have a rich **glossary** of terminology
- Build and maintain the **data dictionary**
  - all nouns in your use-cases are suspect
  - boldface all items in your data dictionary (at least for a consistency check!)
- Use "test balloons" for the customer
  - draw diagrams, suggest scenarios
  - get their attention and keep them interested
  - review and implement team building strategies if needed
  - outside social events, stupid games, etc!
  - See Ludi, Brooks, Yourdon (Death March) and others

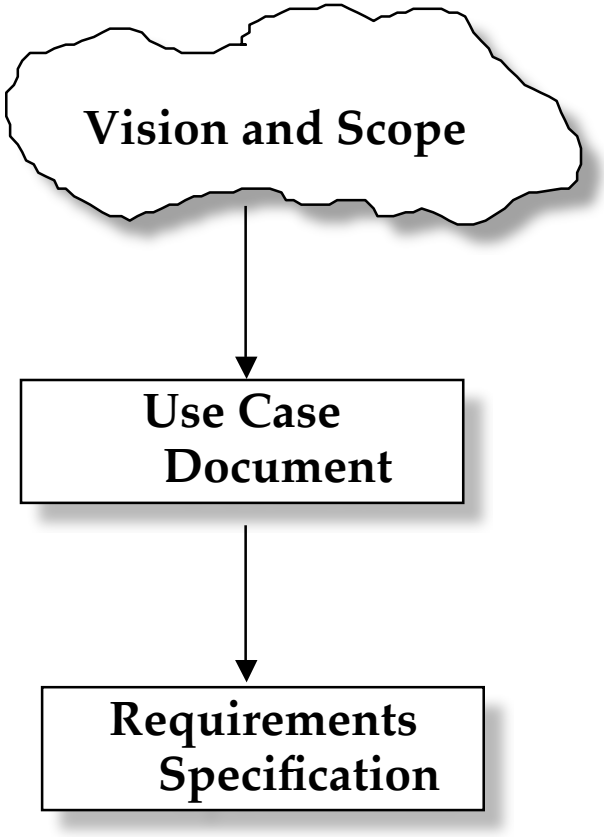
Even More

- Review Schach Chapters, Wiegers Chapter 10
- Pay attention to document QA!
  - team (testing at this stage of the lifecycle)- Document:
    - sanity check
    - English language usage, clarity of language
    - clarity of concepts (or clarity of *questions about* concepts)
- Questions for your user/client/champion
  - consider *who* you address
  - do the questions make clear sense to the recipient?
  - does the question maximize my information "take"?
  - do my question appear professional?
  - am I keeping my customer interested and confident?

More yet again

- Thoughts on Use-Cases
  - review the text, search around, look at examples
  - what are “use-cases” anyway? Really?
  - consider the following qualities
    - ¤ traceability (consistency of usage)
    - ¤ data items (dictionary entries)
    - ¤ glossary terms
    - ¤ use-case table of contents
    - ¤ list of actors, roles, explanation in introduction
    - ¤ descriptions of use-cases: simple, sequential, functional
    - ¤ course of events: model the description graphically

Requirements Analysis and Specification



informal statement of customer's need for system

statement of what users need from system to perform needed tasks

notational and/or formal description of the software system

Purposes for Requirements

- A medium for *communication*
  - between users, developers, coders, testers, managers, marketeers and more
  - must be understandable, answer all relevant questions unambiguously
- A *contract*
  - to prevent (and settle) disputes
  - most common practice: no requirements and "contract" is then expensive to determine

Requirements Analysis continued

- How to facilitate a communications medium?
  - multiple views
    - DFD
    - ERD
    - STD
    - lots more... UML uses all these “old” concepts
  - careful of the tradeoffs: always the cost/benefit analysis to more views
  - some views more suited to certain kinds of projects
    - process oriented (transaction systems) - DFD’s
    - real time systems - STD’s
    - database systems - ERD’s

- Best practice: some combination of views to complement carefully crafted textual requirements
  - UML is an attempt to help us learn this lesson
    - ¤ did OMT do it as well?

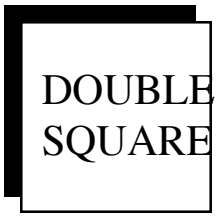
Data-Flow Diagrams

- Captures a system's "logical" data flow
  - Emphasis is on "what" not "how"
    - ¤ Logical: "Records" flow from database to application
    - ¤ Implementation: "Records" are passed via remote procedure call
      - which in turn uses a network protocol called TCP/IP
  - Developed using "stepwise refinement"

Stepwise Refinement

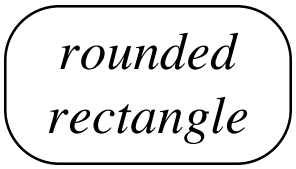
- Stepwise Refinement is a problem-solving technique
- Postpone (requirements/design) decisions on details until as late as possible
- Focus instead on the most important issues
- Miller's Law: 7±2 chunks

Data Flow Diagram Parts

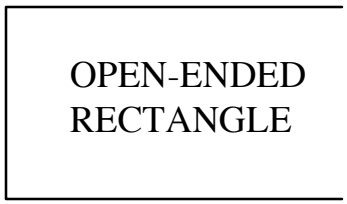


Source or destination of data

Flow of data



Process which transforms a flow of data (note, Wie. uses a circle)



Store of data

Simple Example

- Sally's software shop
  - small shop, buys and resells software at retail
  - sells from stock and makes orders
  - extends credit to inst., corps and some indiv.
  - monthly turnover of 300 packages / $250 each
- Sally says she wants to computerize
  - you ask "why"?
  - what is her goal?
    - ¤ to make more money?
      - more efficiency
      - better customer service

Sally's Software Shop (cont'd)

- ¤ to hide assets from her ex-husband with 9 kids?

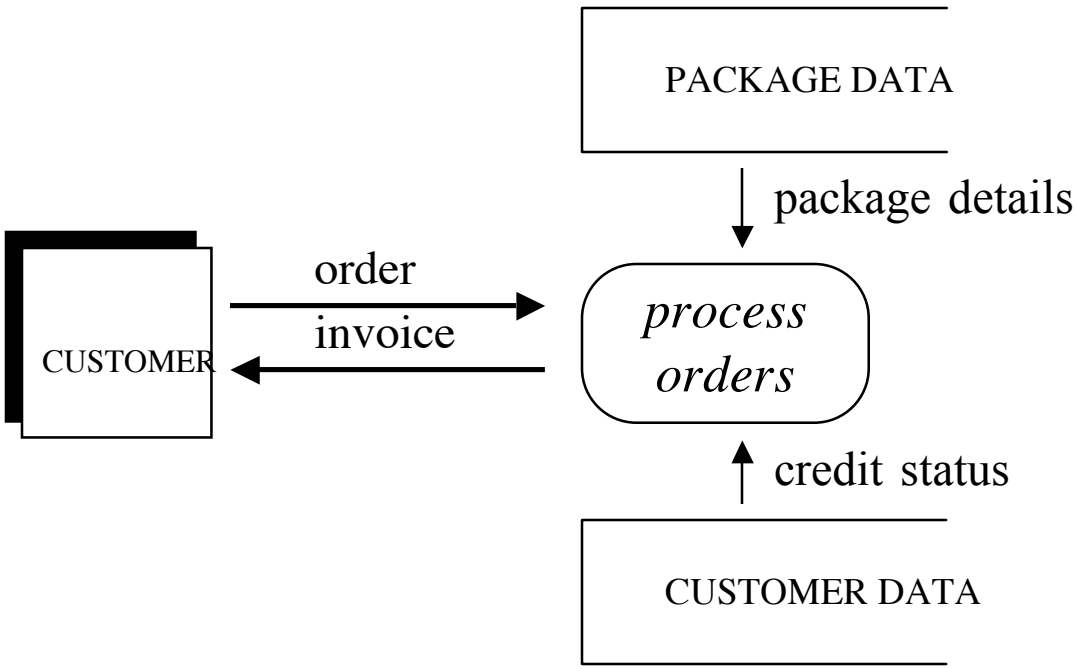
- Do we usually just say "sure" and begin design?
  - find out what the real problem is later...
- Investigate client's needs *first*.
  - DFD's, in addition to use cases help define the problem
- Overall steps (we cover the first):
  - Draw DFD, stepwise refinement
  - Decide what sections to computerize and how (batch or online)
  - Put in details of the data flows
  - Define the logic of the proceses

Sally's Software Shop (cont'd)

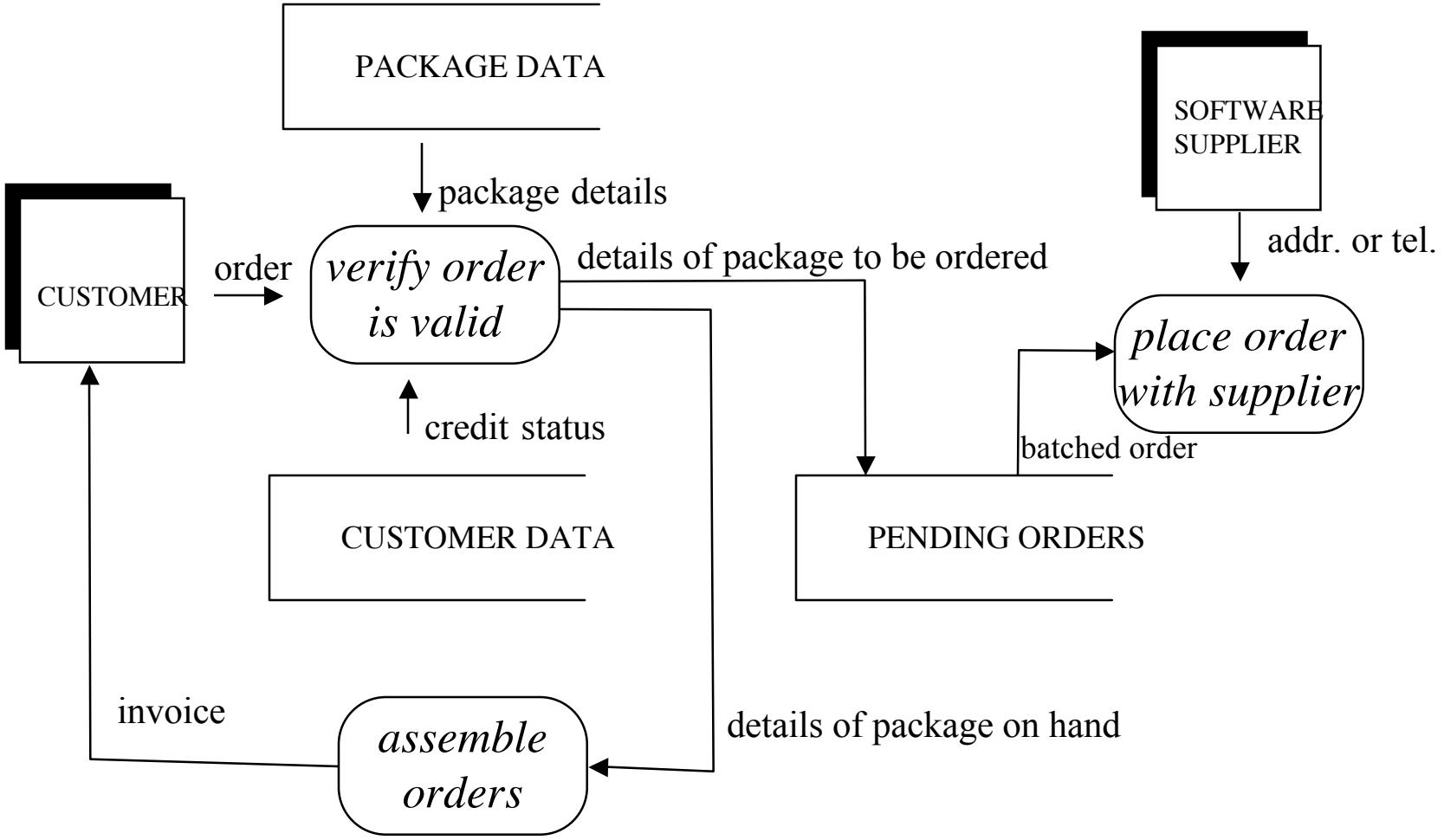
- Define the Data Stores
- Define the Physical Resources
- Determine the Input/ Output Specifications
- Perform Sizing
- Determine the Hardware Requirements

First Step (a context diagram with data stores)

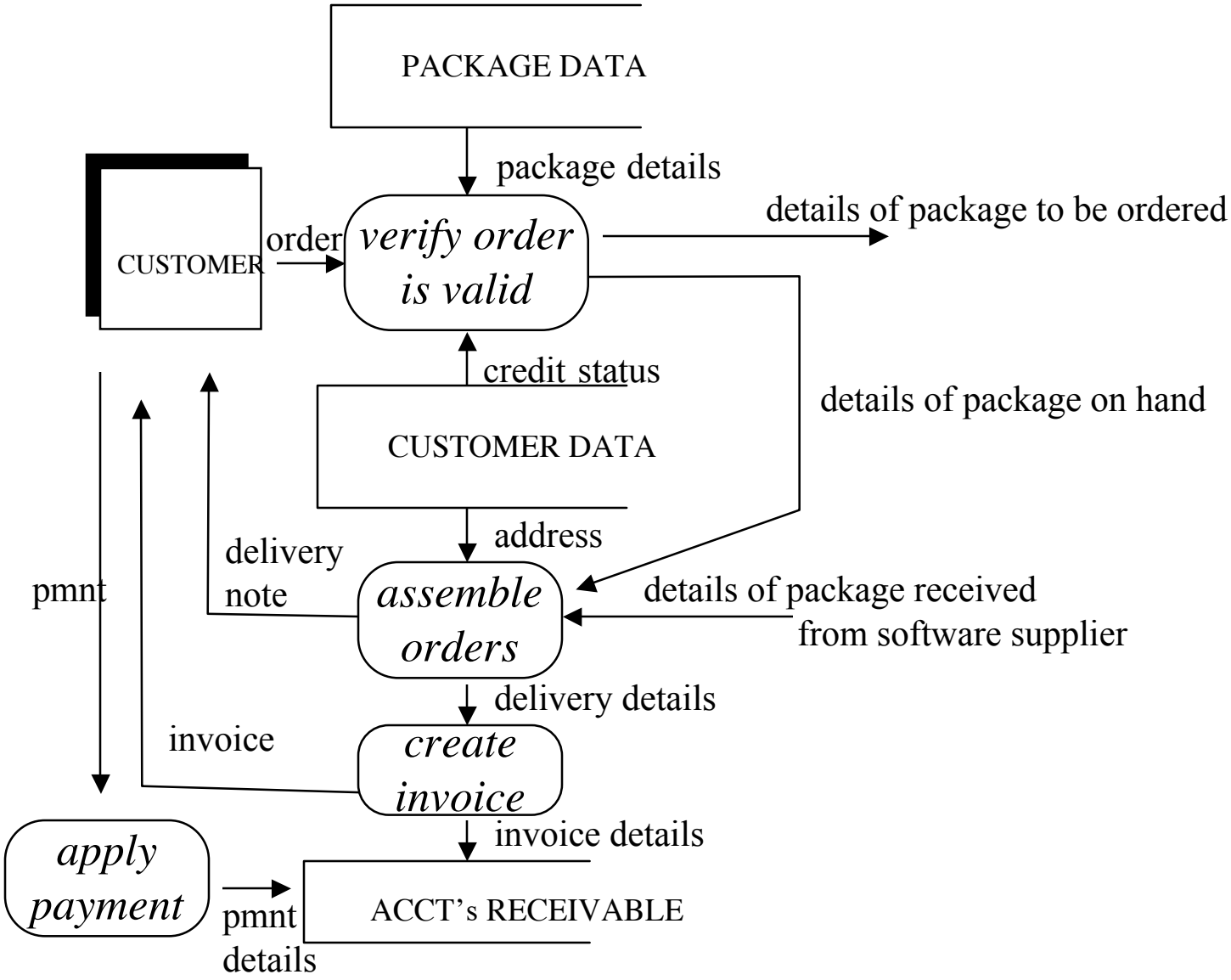
Here are the major players; details on data deferred



Second Refinement



(part of) Third Refinement



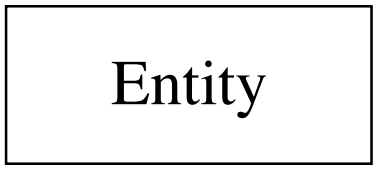
Data-Flow Diagram Wrap-Up

- We could easily continue with our example!
- Determining when to stop depends on your task
  - For requirements, this is a good start
  - For design, we would want to continue on

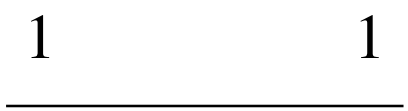
Entity-Relationship Diagrams

- Used to specify relationships between entities in the system
  - These relationships can lend insight into the requirements and/or design of a system
- Originally used in the database domain; now being used in Object-Oriented Analysis

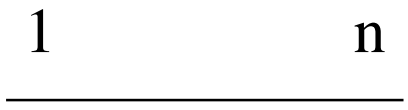
Entity-Relationship Diagram Parts



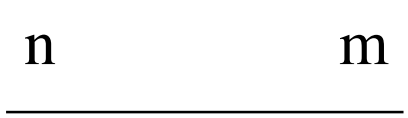
Some object of the system
(look at the Data Dictionary)



one-to-one relationship

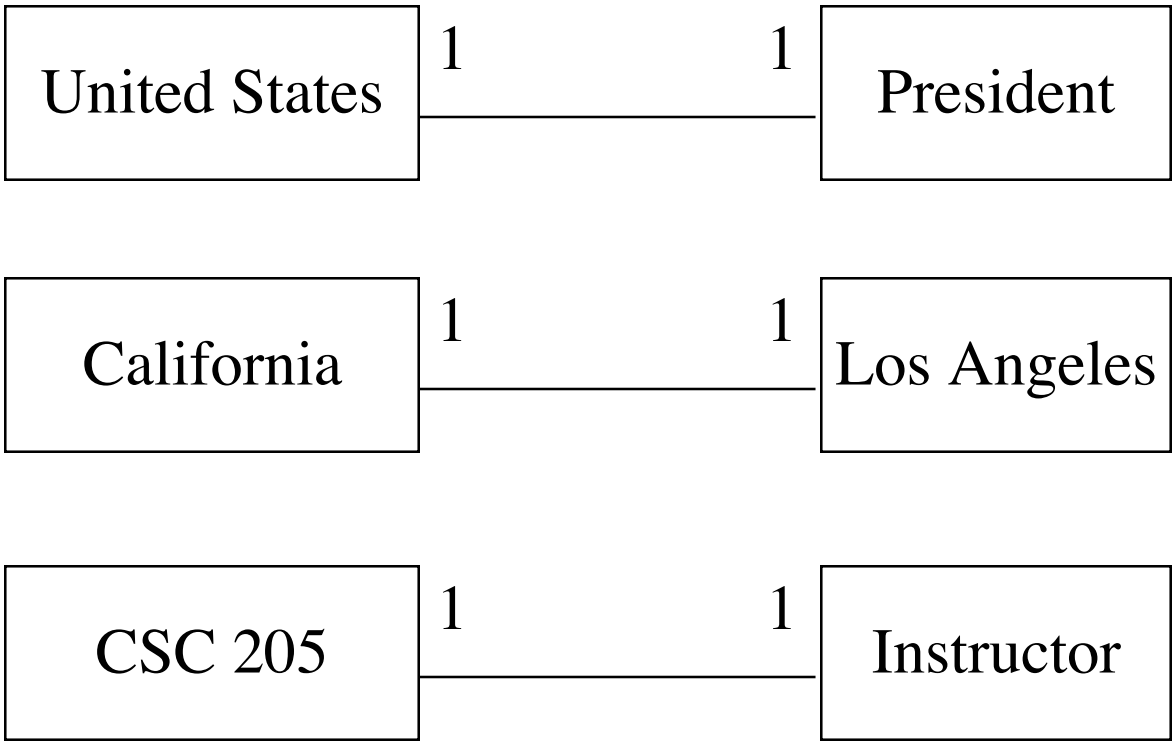


one-to-many relationship

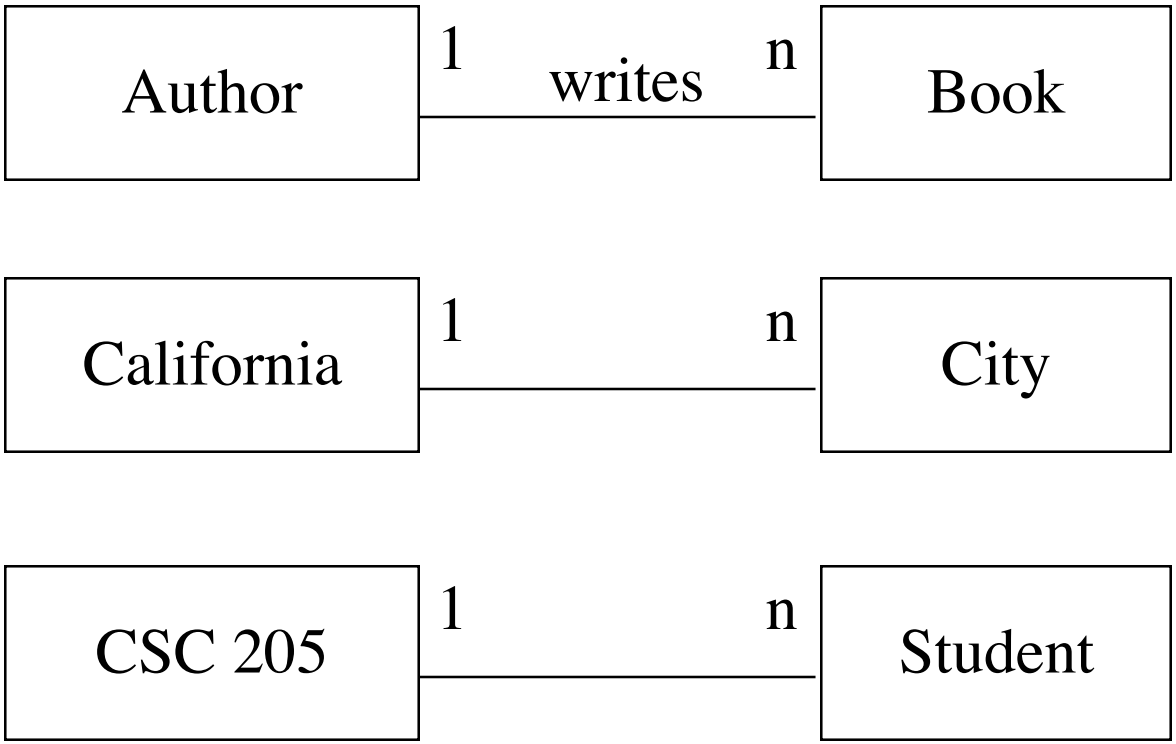


many-to-many relationship

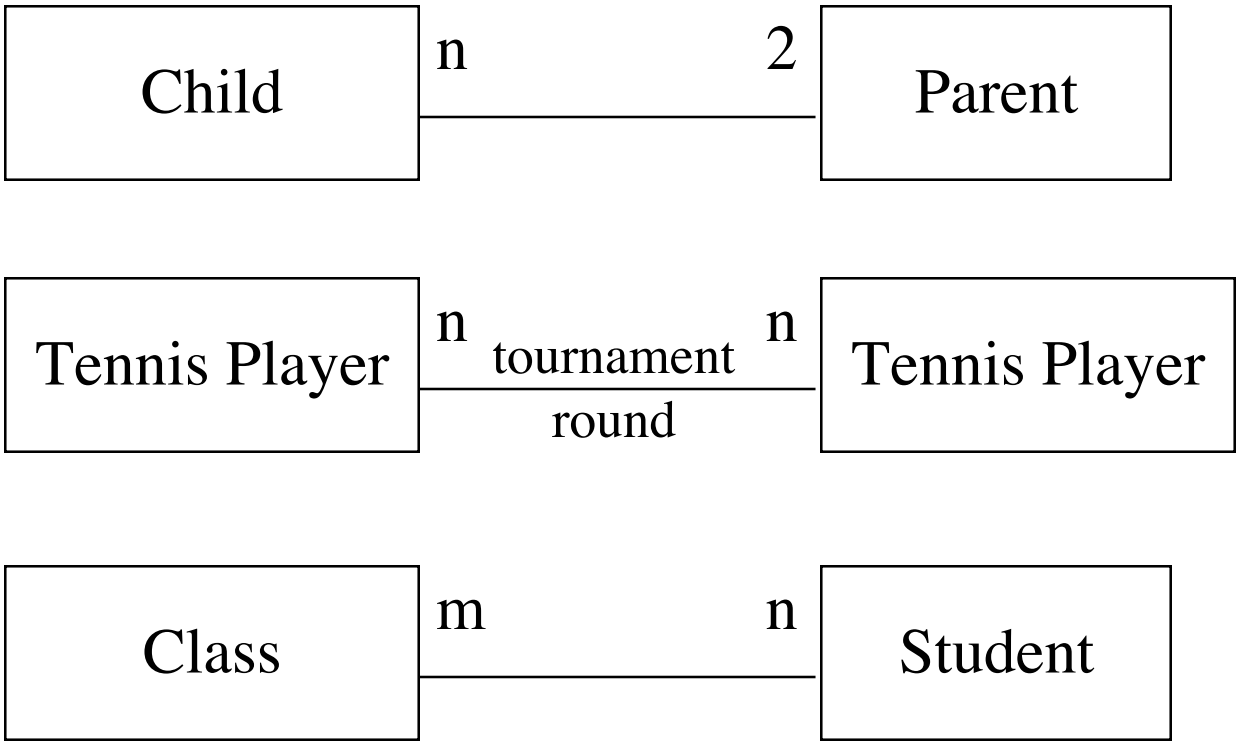
One-to-One Relationships



One-to-Many Relationships



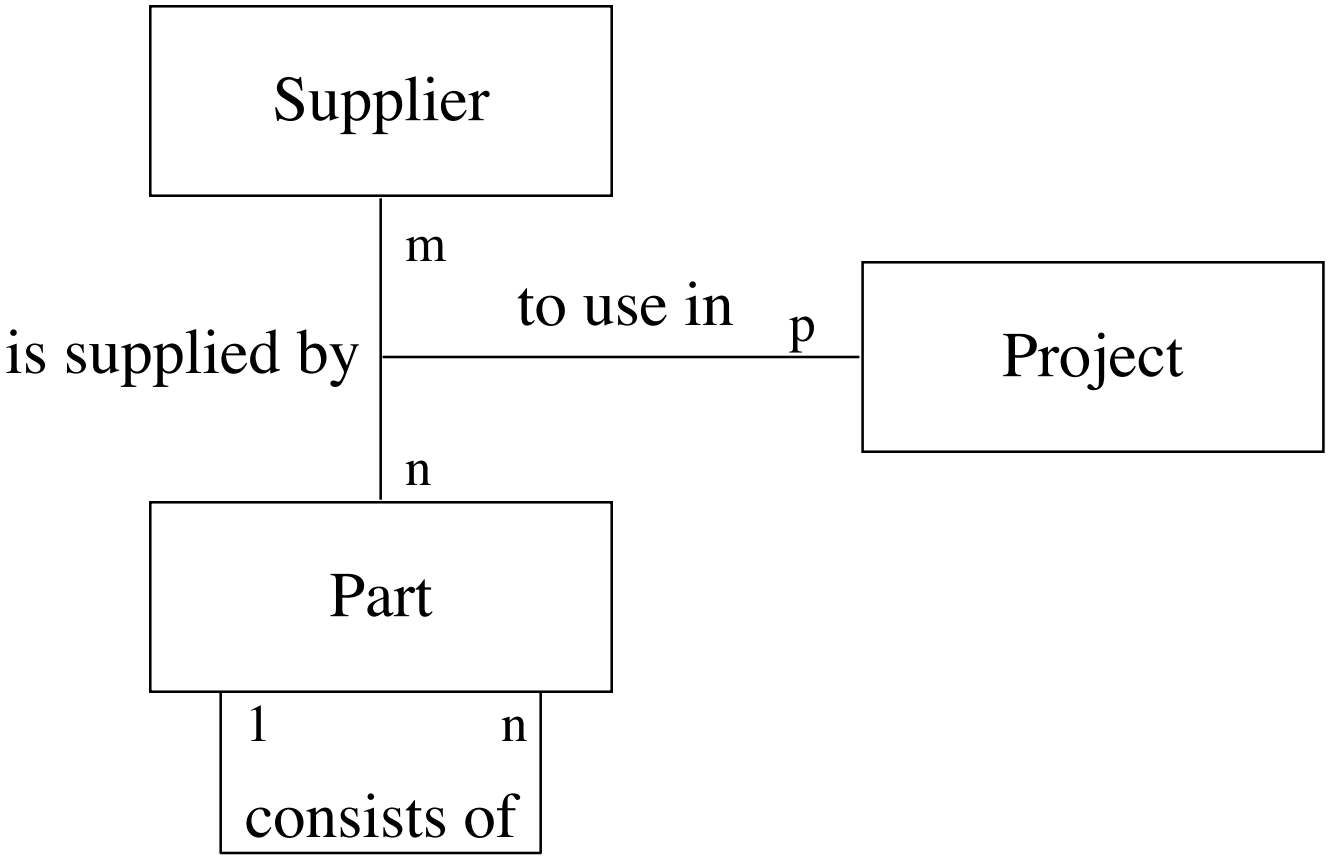
Many-to-Many Relationships



Observations

- Each entity has a name and is singular
  - e.g. “Book” not “Books”, the relationship specifies the number
- Each relationship has a point-of-view, a context, and an optional name.
  - The name helps to establish the context

Example


Supplier
m
is supplied by
to use in
p
Project
n
Part
1
n
consists of

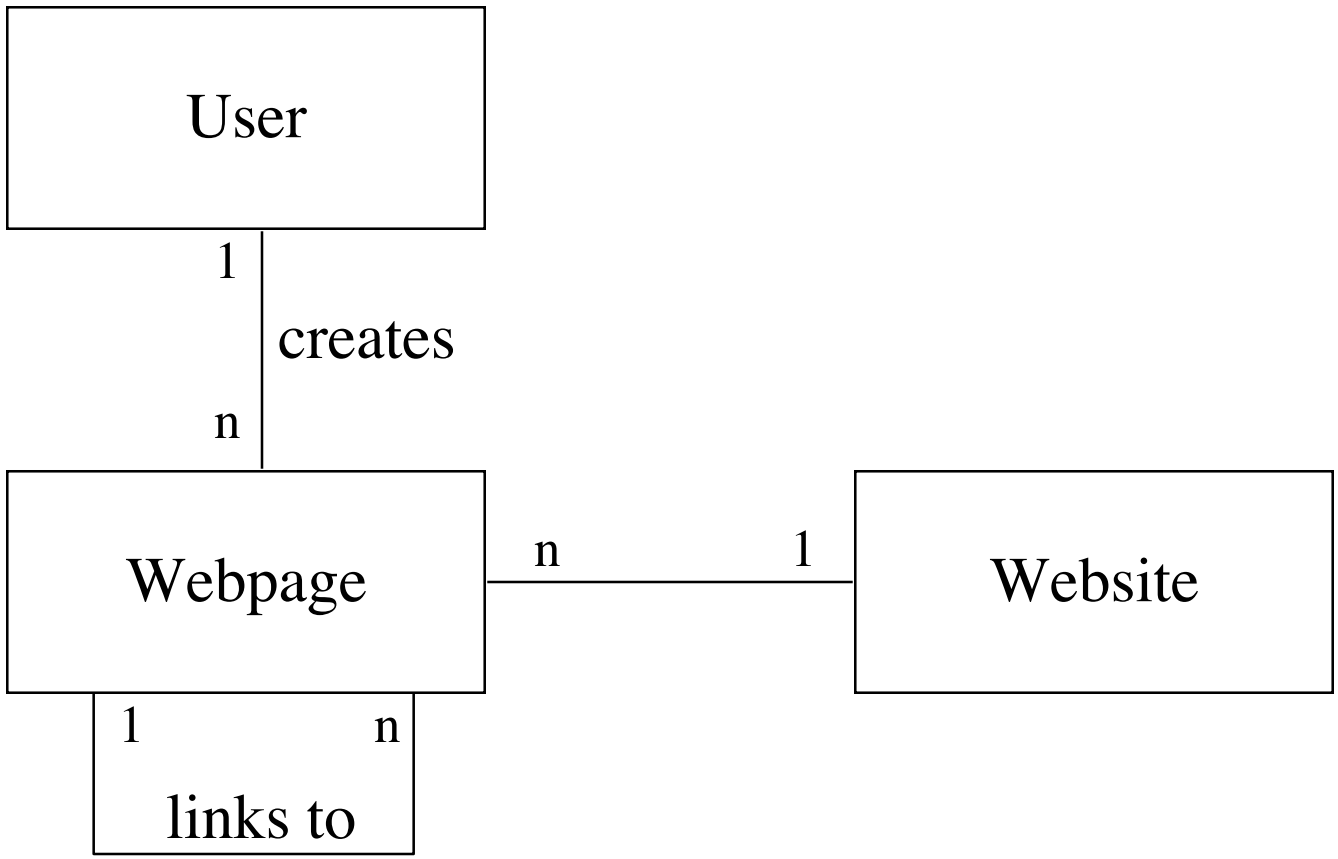
Additional Observations

- An Entity can have relationships with other instances of itself
  - e.g. a part can consist of other parts
- n-ary relationships between entities can be established

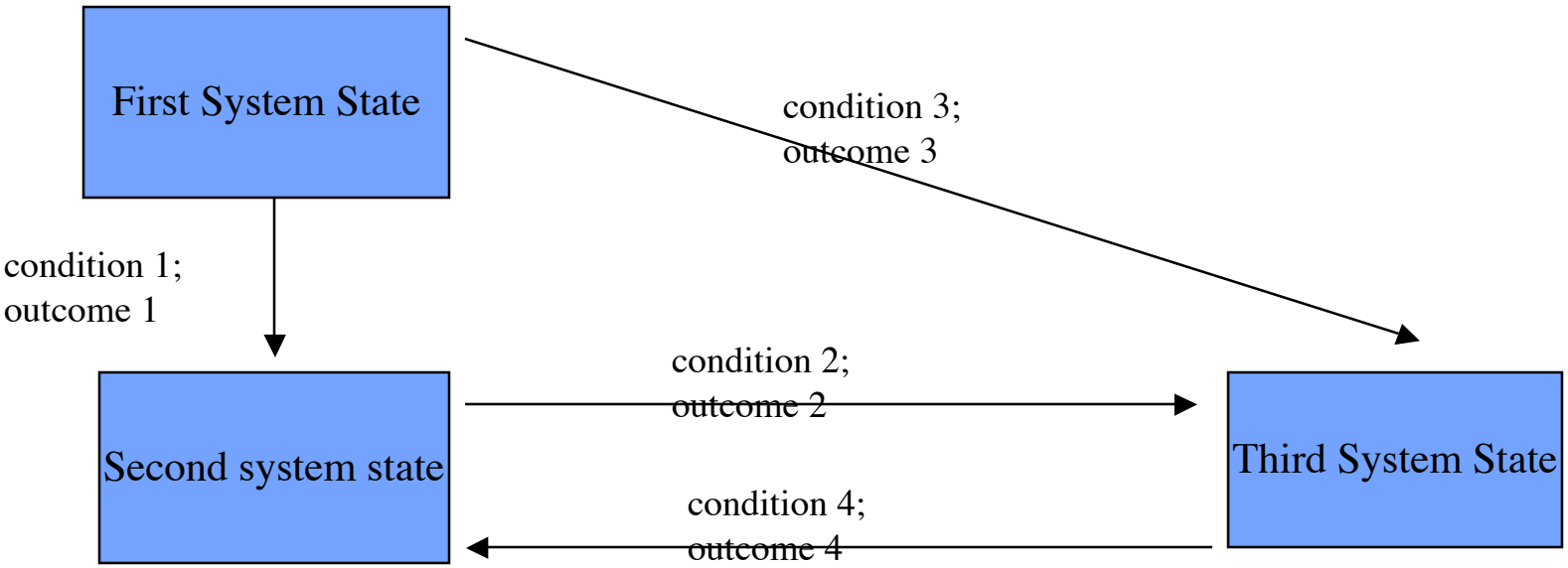
Simple Example

- A user creates many webpages each of which contain many links to other webpages. These webpages are all located on a particular website.

Example Diagram

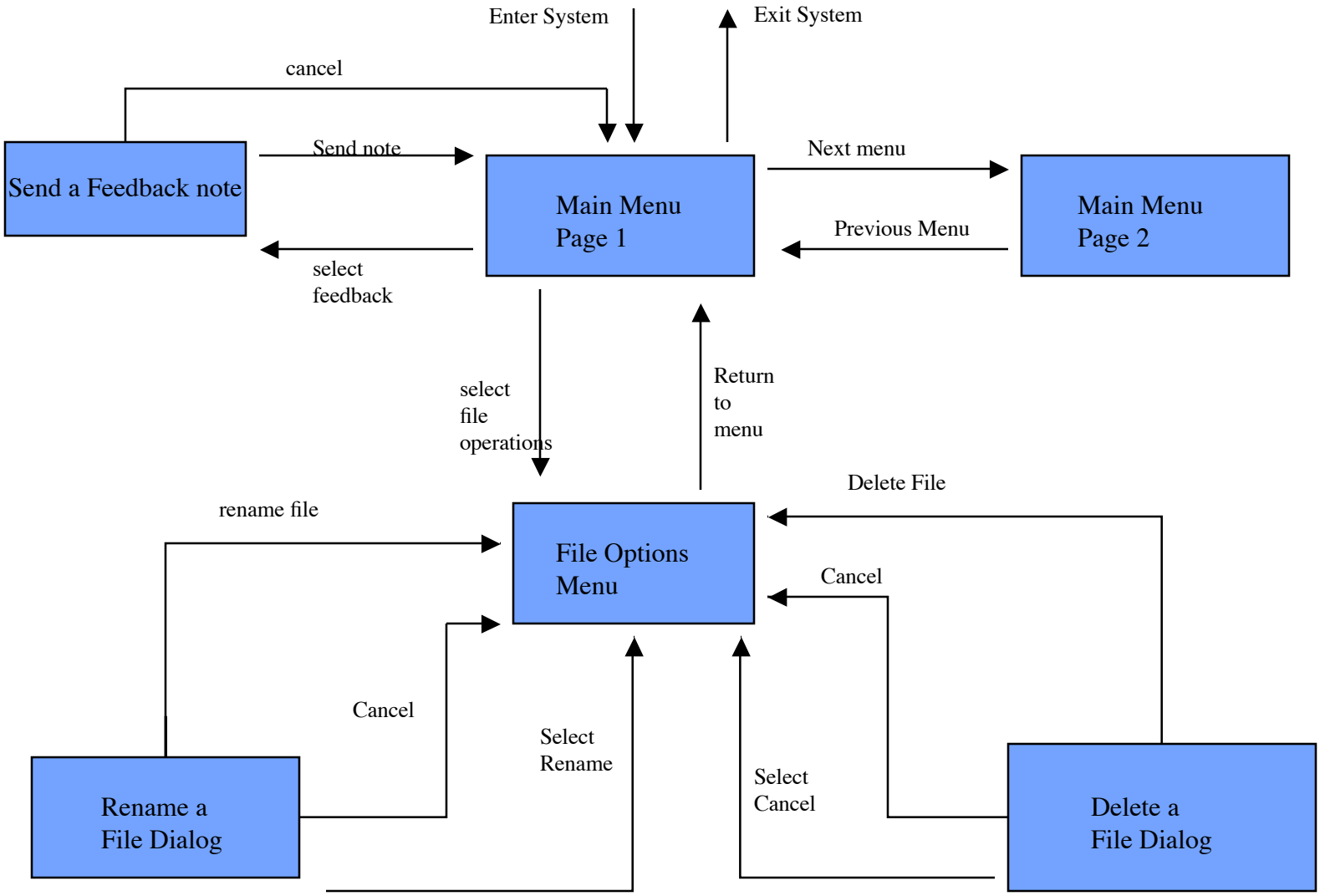


State Transition Diagram (STD)



Models the discrete states a system can be in (state?)
Transitions show the only permitted state changes
Can also model possible status of an object in the system

UI model using a dialog map



Benefits of a dialog map

- Find incorrect or missing transitions early
- Find missing or incorrect requirements early
- Spot opportunities for reuse
- Spot redundancies in UI design
- Can partition the UI into sub components
- Can do state transition diagrams hierarchically, to control the scope of the views

Requirements plus Test Cases plus Dialog Maps

- Requirement: If the stockroom contains smaples of the chemical being ordered, the system shall display a list of the available samples. The user wshall either select a sample, or request to place a new order from the vendor.
- Test Case: At DB40, enter a valid chemical ID with 2 samples in the stockroom. DB50 appears, with the 2 sample numbers. Select sample 2. DB50 closes and sample 2 is added to the bottom of the current chemical request list in DB70.

Dialog Map for Chemical Example