Requirements Definition and Specification

Outline

- Definition and specification
- Natural language
- Semi-formal techniques
  - Program description languages
  - Structured systems analysis
  - Entity-relationship modeling
- Formal techniques
  - Finite state machines
  - Petri nets
- Non-functional requirements

Definition vs. Specification

- Requirement **definitions** are customer-oriented.
  - Written using natural language and easy to understand diagrams.
- Requirement **specifications** are oriented towards the software designer.
  - More precise description of functionality and constraints.

Characteristics of requirements

- Precise
- Complete
- Consistent
- Unambiguous
- Understandable
- Verifiable
  - A set of checks should be defined to verify that the delivered system meets the requirement.

Natural Language Problems

- Lack of Clarity

  It is very difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.

- Requirements confusion

  Functional and non-functional requirements, system goals, and design information may not be clearly distinguished.

- Amalgamation:

  Several different requirements may be expressed as a single requirement.

Example: A CASE tool

2.6 Grid facilities. To assist in the positioning of entities on a diagram, the user may turn on a grid in either centimeters or inches, via an option on the control panel. Initially, the grid is off. The grid may be turned on and off at any time during an editing session and can be toggled between inches and centimeters at any time. A grid option will be provided on the reduce-to-fit view but the number of grid lines shown will be reduced to avoid filling the smaller diagram with grid lines.

Example: Use of standard format

2.6 Grid facilities

2.6.1 **The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window.** This grid shall be a passive grid where the alignment of entities is the user's responsibility.

Rationale: A grid helps the users to … Although an active grid … the user is the best person to decide where entities should be positioned.

2.6.2 When used in 'reduce-to-fit' mode (see 2.1), the number of units separating grid lines must be increased.

Rationale: If line spacing is not increased, the background will be very cluttered with grid lines.

Requirement Specification Methods

- Structured natural language
  - Standard forms, templates, decision tables
- Program description languages
  - Abstract features to specify requirements by defining an operational model
- Requirements specification languages
  - Special purpose languages with tool support
- Graphical notations
- Mathematical specifications

Structured Natural Language

Information should include:

- Description of entity or function being specified
- Inputs (and where they come from)
- Outputs (and where they go)
- What other entities/functions are used
- Pre-conditions and post-conditions (if functional approach is used)
- Side-effects

Program description languages

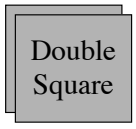
- Describe requirements *operationally*
- Derived from programming languages like Ada
- May contain additional, more abstract constructs

```
procedure ATM is
   PIN: Pin_no;
   Balance: Amount;
   Service: Available_services;
   Valid_card, Valid_PIN: Boolean;
begin
   loop
      Get_card(Acc_no, PIN, Valid_card);
      if Valid_card then
         Validate_PIN(PIN, Valid_PIN);
         if Valid_PIN then
            Get_account(Acc_no, Balance);
            Get_service(Service);
            while a service is selected loop
               Deliver_selected_service;
               Get_service(Service);
```

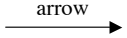
PDL's (2)

- Advantages
  - May be checked syntactically and semantically be software tools
  - Descriptions of sequences of actions in natural language are often confusing. May be used in conjunction with natural language when more detail is required.
  - PDL allows more detail about interfaces between sub-systems which are often defined in the requirements (particularly if some sub-systems already exist)
  - More natural transition from requirements to design
- Disadvantages
  - May not be expressive enough to describe problem domain concepts
  - Design decisions may be made too early

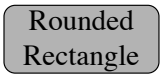
Structured systems analysis



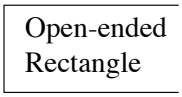
Source or destination of data



Flow of data



Process which transforms a flow of data

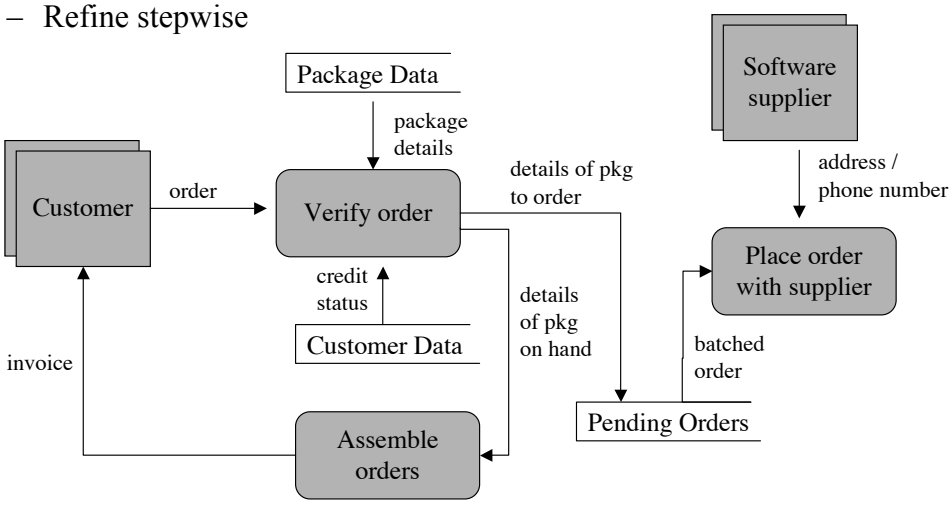


Store of data

Structured systems analysis (2)

Example: Computerizing Sally's Software Shop

- Step 1. Draw the data flow diagram (DFD).
  - Refine stepwise



Structured systems analysis (3)

- Step 2. Decide what sections to computerize and how (e.g. batch or online).
- Step 3. Put in the details of the data flows.

  Order:

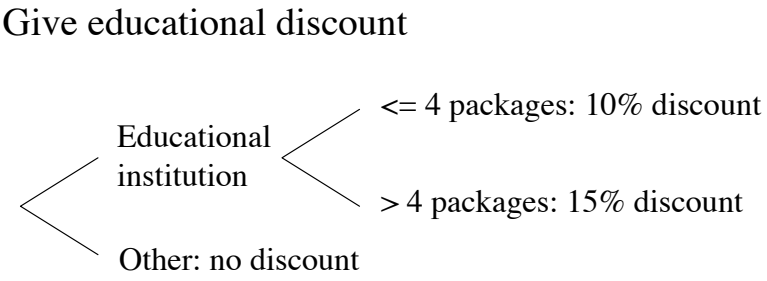
  Order identification

  Customer details

  Package details
  - Continue stepwise refinement

Structured systems analysis (4)

- Step 4. Define the logic of the processes.
  - Details of what happens within each process
  - Use decision trees where appropriate



Structured systems analysis (5)

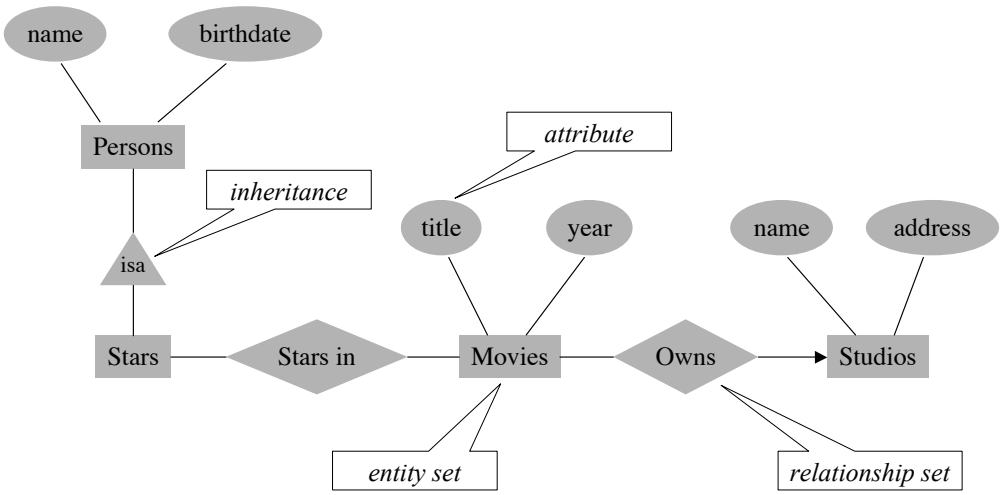
- Step 5. Define the data stores.
  - Exact contents and representation
- Step 6. Define the physical resources.
  - File names, organization, storage media, etc.
  - Or, the DBMS to be used and database schema
- Step 7. Determine input/output specifications.
  - Format of input, user screens, etc.
  - Output format

Structured systems analysis (6)

- Step 8. Perform sizing.
  - Size of inputs and outputs
  - Frequency of inputs and reports, report deadlines
  - Size and number of records that are transferred between primary and secondary storage
- Step 9. Determine the hardware requirements.

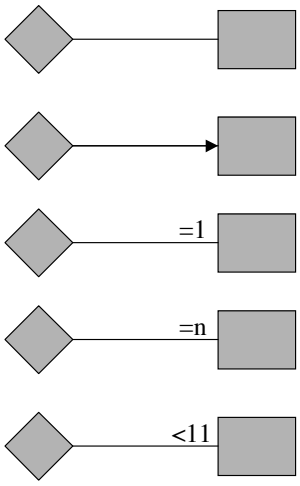
Entity-relationship Modeling

- Emphasis on data rather than processes



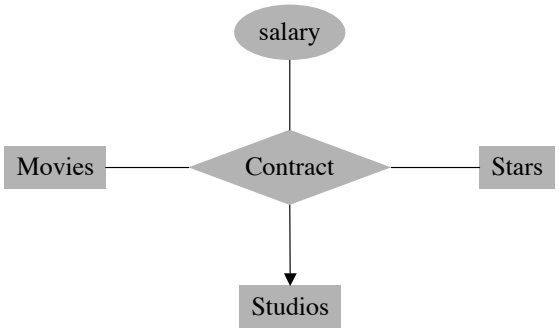
E/R Modeling (2)

- Multiplicities of relationships



E/R Modeling (3)

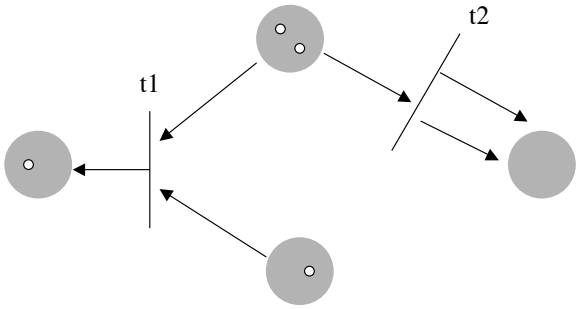
- Relationships involving more than two entities



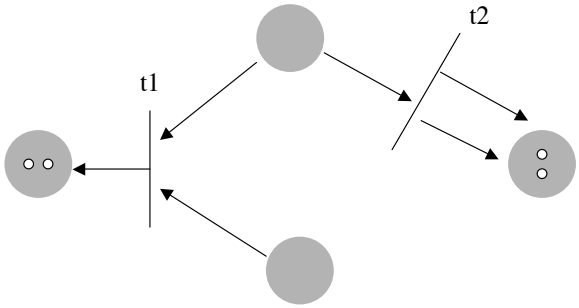
Petri nets

- Useful for describing concurrent activities
- Petri nets consist of:
  - A set of places
  - A set of transitions
  - An input function (maps transitions to places)
  - An output function (maps transitions to places)
- A marking is an assignment of tokens
- Input and output functions are shown as arcs
- A transition can fire when for each of its input arcs (and none of its inhibitor arcs) there is a token for the source of the arc
- When a transition fires, tokens are placed at the target of each output arcs

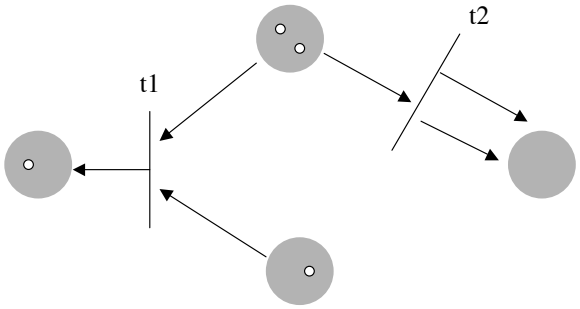
Petri nets (2)



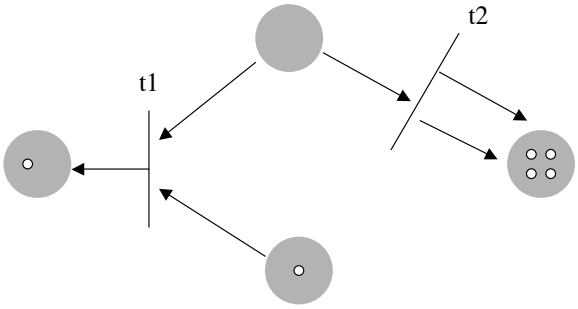
After t1 and t2 fire



Petri nets (3)

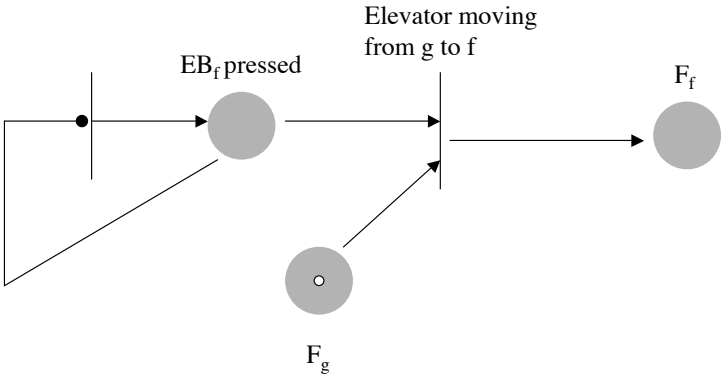


After t2 fires twice



Petri nets (4)

Elevator buttons



Non-functional requirements