**Last Week:**
Elicitation Techniques
- Contextual approaches
- Cognitive approaches

**This Week:**
- Modeling Enterprises
  - General Modeling Issues
  - Modeling Human Activity
  - Decomposition, Means-Ends Analysis
  - and task dependencies

**Next Week:**
- Modeling Info and Behaviour
- Structured and OO methods
- ER and Class Hierarchies
- State machines

---

**Lecture 5: Modeling Enterprises**

*University of Toronto*  
*Department of Computer Science*

**Motivation for enterprise modeling...**
Imagine we have interviewed some stakeholders...

<table>
<thead>
<tr>
<th>Chief Executive</th>
<th>Catering Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>When flights are full VIPs are first to be upgraded.</td>
<td>The food loaded is dictated by the number of passengers travelling in a particular class.</td>
</tr>
<tr>
<td>Discounted tickets should be offered to politicians since they make important decisions affecting the airline.</td>
<td>A predicted number of passengers on a flight must be available 24 hours prior to departure.</td>
</tr>
<tr>
<td>Info about frequent fliers should not be made available to outside contractors.</td>
<td>Passengers requiring special needs must indicate their request 24 hours prior to departure.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chief Security Officer</th>
<th>Airline Sales manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of bags in the aircraft’s hold should tally against the list of passengers on board.</td>
<td>A ticket may only be issued when a fare is paid.</td>
</tr>
<tr>
<td>For some fares, a reservation can be held and not confirmed.</td>
<td>When a discounted ticket is booked, the normal book-ahead requirements do not apply.</td>
</tr>
<tr>
<td>Passenger lists should not be made available to the public.</td>
<td>All tickets must carry appropriate endorsements relating to the terms and conditions of issue of tickets.</td>
</tr>
</tbody>
</table>

**How do we get from here to an agreed specification?**

**A model is more than just a description**

*it has its own phenomena, and its own relationships among those phenomena.*

*The model is only useful if the model’s phenomena correspond in a systematic way to the phenomena of the domain being modelled.*

*Example:* Bio: entity  
Person: entity  
Phenomena in the application domain that are not in the model  
Phenomena in the model that are not present in the application domain

**Remember: “It’s only a model”**

*There will always be:*  
- phenomena in the model that are not present in the application domain  
- phenomena in the application domain that are not in the model

*“If the map and the terrain disagree, believe the terrain”*  
Perfecting the model is not always a good use of your time...

---

*Source: Adapted from Jackson, 2001, p.120-123*
Modeling...

→ Modeling can guide elicitation:
  % Does the modeling process help you figure out what questions to ask?
  % Does the modeling process help to surface hidden requirements?
  % i.e. does it help you ask the right questions?

→ Modeling can provide a measure of progress:
  % Does completeness of the model imply completeness of the elicitation?
  % i.e. if we’ve filled in all the pieces of the model, are we done?

→ Modeling can help to uncover problems
  % Does inconsistency in the model reveal interesting things?
  % e.g. inconsistency could correspond to conflicting or infeasible requirements
  % e.g. inconsistency could mean confusion over terminology, scope, etc
  % e.g. inconsistency could reveal disagreements between stakeholders

→ Modeling can help us check our understanding
  % Can we test that the model has the properties we expect?
  % Can we reason over the model to understand its consequences?
  % Can we animate the model to help us visualize/validate the requirements?

Type of Model

Can choose a variety of conceptual schema:

→ natural language
  % extremely expressive and flexible
  % very poor at capturing the semantics of the model
  % better used for elicitation, and to annotate models for communication

→ semi-formal notation
  % captures structure and some semantics
  % can perform (some) reasoning, consistency checking, animation, etc.
  % e.g. diagrams, tables, structured English, etc.
  % mostly visual – for rapid communication with a variety of stakeholders

→ formal notation
  % very precise semantics, extensive reasoning possible
  % long way removed from the application domain
  % note: requirements formalisms are geared towards cognitive considerations, hence differ from most computer science formalisms

Desiderata for Conceptual Schema

→ Implementation Independence
  % does not model data representation, internal organization, etc.

→ Abstraction
  % extracts essential aspects
  % e.g. things not subject to frequent change

→ Formality
  % unambiguous syntax
  % rich semantic theory

→ Constructability
  % can construct pieces of the model to handle complexity and size
  % construction should facilitate communication

→ Ease of analysis
  % ability to analyze for ambiguity, incompleteness, inconsistency

→ Traceability
  % ability to cross-reference elements
  % ability to link to design, implementation, etc.

→ Executability
  % can animate the model, to compare it to reality

→ Minimality
  % No redundancy of concepts in the modeling scheme
  % i.e. no extraneous choices of how to represent something

Meta-Modeling

→ Can compare modeling schema using meta-models:
  % What phenomena does each scheme capture?
  % What guidance is there for how to elaborate the models?
  % What analysis can be performed on the models?

→ Example meta-model:

```
States: Facts, Events
Actions: modify, record
Activities: trigger
Propositions about the application domain
```

Source: Adapted from Loucopoulos & Karakostas, 1995, p23-24
Modeling Principle 1: Partitioning

→ Partitioning
% captures aggregation/part-of relationship

→ Example:
% goal is to develop a spacecraft
% partition the problem into parts:
  > guidance and navigation;
  > data handling;
  > command and control;
  > environmental control;
  > instrumentation;
  > etc.
% Note: this is not a design, it is a problem decomposition
% actual design might have any number of components, with no relation to these sub-problems
% However, the choice of problem decomposition will probably be reflected in the design

Modeling Principle 2: Abstraction

→ Abstraction
% A way of finding similarities between concepts by ignoring some details
% Focuses on the general/specific relationship between phenomena
  > Classification groups entities with a similar role as members of a single class
  > Generalization expresses similarities between different classes in an 'is_a' association

→ Example:
% requirement is to handle faults on the spacecraft
% might group different faults into fault classes based on location:
  > instrumentation fault,
  > communication fault,
  > processor fault,
  > etc
% based on symptoms:
  > no response from device;
  > incorrect response;
  > self-test failure;
  > etc...

Modeling Principle 3: Projection

→ Projection:
% separates aspects of the model into multiple viewpoints
  > similar to projections used by architects for buildings

→ Example:
% Need to model the communication between spacecraft and ground system
% Model separately:
  > sequencing of messages;
  > format of data packets;
  > error correction behavior;
  > etc.

→ Note:
% Projection and Partitioning are similar:
  > Partitioning defines a 'part of' relationship
  > Projection defines a 'view of' relationship
% Partitioning assumes the parts are relatively independent

Survey of Modeling Techniques

→ Modeling Enterprises
% Goals & objectives
% Organizational structure
% Tasks & dependencies
% Agents, roles, intentionality

→ Modeling Information & Behaviour
% Information Structure
% Behavioral views
  > Scenarios and Use Cases
  > State machine models
  > Information flow
% Timing/Sequencing requirements

→ Modeling System Qualities (NFRs)
% Product requirements
% Process requirements
% External requirements

Source: Adapted from Davis, 1990, p48 and Loucopoulos & Karakostas, 1995, p78.
Approaches to Enterprise Modeling

→ 1970's
  - Soft Systems Approaches:
    - Involve the entire organization
    - Be sensitive to political and social context for organisational change
  - Examples: SSM, ISAC

→ 1980's
  - Knowledge-based Approaches:
    - Use knowledge representation schemes to build executable domain models
    - Capture static and dynamic aspects of the domain
  - Examples: RML, Requirements Apprentice, Nature

→ 1990's
  - Teleological Approaches:
    - Requirements are really just goals, so model goal hierarchies
    - Focus on the 'why' question, rather than 'what/how'
    - And use scenarios as concrete examples of how goals are (can be) satisfied
  - Examples: KAOS, i*, CREWS, ...

→ 2000's ...

ISAC

Information Systems Work & Analysis of Changes (ISAC)

- Developed in the 1970's in Sweden
- Emphasizes cooperation between users, developers and sponsors
  - Developers' role is to facilitate the process
- Good for information systems; not applicable to control systems.

ISAC Process

1. Change Analysis
   - What does the organization want?
   - How flexible is the organization with respect to changes?
2. Activity Study
   - Which activities should we regroup into information systems?
   - Which priorities do the information systems have?
3. Information Analysis
   - Which inputs and outputs do each information system have?
   - What are the quantitative requirements on each information system?
4. Implementation
   - Which technology (info carriers; h/w; s/w) do we use for the information systems?
   - Which activities of each information system are manual, which automatic?

ISAC Change Analysis

1. List problems
   - Dissatisfactions with current system
     - List all problems
     - Then remove any that are trivial or intractable

2. List interest groups
   - These are "problem owners"
   - Draw matrix of problems against owners
     - This exercise is done with the problem owner’s involvement

3. Analyze problems
   - Use cause-effect analysis
     - Identify solution-oriented problems, to get to underlying causes
   - Performed by domain specialists
   - Quantify the problems

4. Make Current Activity Model
   - Notation: A-schemas (similar to dataflow diagrams)

5. Analyze Goals
   - Declarative statement of goals
     - I.e. desired result, not how to get there
   - Result should be a tree of goals

6. Define Change Needs
   - Goals should explain why the problems exist; problems frustrate goals
   - Cluster problems into related groups
     - Each group is a change need

7. Generate Change Alternatives

8. Model desired situations
   - Make packages of change alternatives

9. Evaluate Alternatives

10. Choose an alternative

Soft System Methodology (SSM)

→ Background
  - Developed by Checkland in late 1970's
  - Reality is socially constructed, and therefore requirements are not objective
  - Rationale:
    - Problem situations are fuzzy (not structured) and solutions not readily apparent.
    - Impact of a computerization may be negative (e.g. intro of new system reduced productivity as it removed employee motivation)
    - Full exploitation of computerization may need radical restructuring of work processes.

→ Approach
  - Analyze problem situation using different viewpoints
    - Determining the requirements is a discussion, bargaining and construction process.
  - Out of this process emerges not just a specification, but also:
    - Plan for a modified organization structure
    - Task structures
    - Objectives
    - Understanding of the environment
SSM Approach

1 Existing situation (unstructured problem)
2 Analyze the problem situation
   - Draw a rich picture
   - Look for problem themes (describe them in natural language)
3 Define relevant systems and root definitions (CATWOE)
   - A root definition is a concise description of a human activity system
4 Build a conceptual model
   - Of the activity system needed to achieve the transformation
   - Process oriented model, with activities & flow of resources
5 Compare conceptual model with step (2)
   - Ordered questioning - questions based on the model
   - Event reconstruction - take past events and compare them to the model
   - General comparison - look for features of the model that are different from current situation
   - Root definition - point by point comparison of the two models
6 Debate feasible and desirable changes
   Three types of change: structural, procedural, attitudinal
7 Implement changes

SSM Modeling

Root definition:
"A hospital-owned system, which provides records of spending on drugs so that control action by administrators and doctors to meet defined budgets can be taken jointly"

Customers: Administrators, Doctors
A: Not stated
Transformation: Need to know spending on drugs
Need met by recording info.
W: Monitoring spending on drugs is possible and is an adequate basis for joint control action
O: Hospital
E: Hospital mechanisms, roles of administrators and doctors, defined budgets

SSM modeling

E.g. Strategic Dependency Model

- Shows means-ends links between tasks and goals
- SR model shows interactions between goals within each actor
- Shows task decompositions

Legend
- D: Direct Dependency
- T: Task Dependency
- G: Goal Dependency
- N: Neural Dependency
- O: Open Unconnected
- X: Critical

This diagram ©2001, Eric Yu
KAOS

→ Background
- Developed in the early 90’s
  - First major teleological requirements modeling language
  - Full tool support available
  - Has been applied to a number of industrial case studies
- Two parts:
  - Informal goal structuring model
  - Formal definitions for each entry in temporal logic

→ Approach
- Method focuses on goal elaboration:
  - Define initial set of high level goals & objects they refer to
  - Define initial set of agents and actions they are capable of
- Then iteratively:
  - Refine goals using AND/OR decomposition
  - Identify obstacles to goals, and goal conflicts
  - Operationalize goals into constraints that can be assigned to individual agents
  - Refine & formalize definitions of objects & actions

Using UML for enterprise modelling

→ Use Cases
- Already assume the basic functions of the machine have been decided
- Hence, it’s premature to look for Use Cases yet...

→ Collaboration/Activity Diagrams
- Show how classes (actors?) collaborate to perform tasks
  - Represent “message” flows between objects
- Offer a simple way of diagramming scenarios
- But do not show:
  - Intentionality, task dependency, task decomposition

→ Class diagrams
- Show the actors/roles and entities in the domain
- Concentrate on static structure
- Can implicitly capture business rules through multiplicity constraints
- Must remember to model domain entities rather than implementation classes

→ Conclusion
- UML offers only very crude tools for enterprise modeling
Collaboration Diagrams

→ Example - “select courses to teach”

1: Inform(courseList)  
Professor  

2: “[for each professor] Inform(courseList)  
AssociateChair  
Professor

3: Propose(courseList)  
4: Agree(courseList)  
5: Update(courseList)  
6: “[For each course] Update()  
7: “[For each professor] Update()

→ Note:  
% Impossible to tell whether this is an indicative or optative description

Example Activity Diagram

Receive Order  
[for each line item on order]  
[failed]  
Authorize Payment  
Check Line Item  
[need to reorder]  
Cancel Order  
[failed]  
Assign to Order  
[stock assigned to all line items and payment authorized]  
Dispatch Order  
Reorder Item  
[all outstanding order items filled]  
Add Remainder to Stock

Activity Diagram with Swimlanes

Finance  
Receive Order  
[for each line item on order]  
[failed]  
Authorize Payment  
[succeeded]  
Cancel Order  
[need to reorder]  
Dispatch Order  
[all outstanding order items filled]  
Add Remainder to Stock

Order Processing  
Receive Supply  
Check Line Item  
[in stock]  
Choose Outstanding Order Items  
Assign Goods to Order  
Assign to Order

Stock Manager  

Class Associations

Multiplicity  
A client has exactly one staff member as a contact person

Name of the association  

Role  
The staff member’s role in this association is as a contact person

Direction  
The “liaises with” association should be read in this direction

Role  
The client’s role in this association is as a clientList
Capturing Business Rules

→ Why do we care about business rules?
  % They help us to understand the business context
  % They could be important constraints for the design of the new system
  > E.g. constraints on when and how operations can happen
  > E.g. constraints on the state space of objects
  % They help us write "operation specifications" for class operations

→ How do we specify business rules?
  % Natural Language
    > such descriptions can be highly ambiguous
  % Structured English
    > use a subset of a natural language (limited syntax and vocabulary)
    > can be hard to write, hard to verify, and too close to program code
  % Decision Tables
    > use a table representation of alternative outcomes (similar to truth tables)
  % Decision Trees
    > use a tree representation of alternative outcomes
  % Object Constraint Language
    > UML notation for adding extra constraints to models
    > Can also be used for specifying pre- and post-conditions on operations

Decision Tables

→ Inputs as columns, actions (outputs) as rows
  % If there are n parameters (conditions) to a decision, each with k₁, k₂,...,kₙ
  % values, then table has:
    > k₁ x k₂ x ... x kₙ columns
    > as many rows as there are possible actions
  % For example:
    > "If the plane is more than half full and the flight costs more than $350 per seat, serve free cocktails, unless it is a domestic flight. Charge for cocktails in all domestic flights where cocktails are served, i.e., those that are more than half full"

<table>
<thead>
<tr>
<th>conditions</th>
<th>Domestic?</th>
<th>$350/seat</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥ half full?</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Serve cocktails?</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Free cocktails?</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Decision Trees

→ Represent the decision logic as a tree:
  % Nodes of the tree represent input parameters (questions)
  % Leaf nodes of the tree represent outputs (actions)

→ Example:

```
Short Trip?
  /    |
/     |
Have Car?
  /    |
Have Car? on a budget?
  /    |
Take Car
  /    |
Walk
  /    |
Taxi

In-Town Trip?
  /    |
/     |
Have Car?
  /    |
Have Car? on a budget?
  /    |
Take Car
  /    |
TTC
  /    |
Taxi

Out-of-Town Trip?
  /    |
/     |
Have Car?
  /    |
Fly
  /    |
```