II. Conceptual Modeling

Engineering Software
Models in Software Engineering
What is Conceptual Modeling?
Origins

Engineering Software

- All engineering disciplines are founded on models that are analyzable and can predict the properties of the artifact being engineered.
- Examples: Electric circuits, bridges, car engines.
- What kinds of models can we build for information systems?… And how do we analyze them?
- We will look at modeling and analysis techniques for requirements and for designs.

Conceptual Modeling

- Key problem: Have to give an unambiguous, easy to understand account of our understanding of an organization and how it works, also how the new system will fit in that organization.
- We can do so with English descriptions; but such descriptions are often cumbersome, incomplete, ambiguous and can lead to misunderstandings (…see next two slides!)
- As an alternative, we will use ***conceptual models*** (also called ***visual models***) to describe proposed requirements and designs for the new system.
- Conceptual models (try to) capture people's understanding (conceptualization) of what is being modeled.
- Conceptual models are usually represented in terms of a graph structure.

Natural Languages can be Ambiguous

- This is clause 4 from the UN Security Council resolution 1441: [on Iraq]

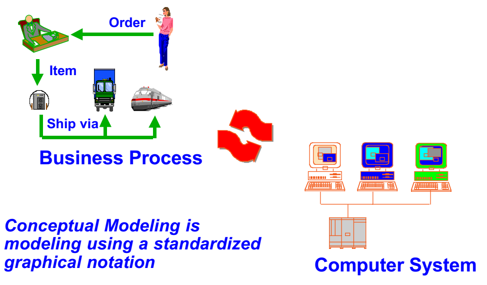
"Decides that false statements ***or*** omissions in the declarations submitted by Iraq pursuant to this resolution ***and*** failure by Iraq at any time to comply with, ***and*** cooperate fully in the implementation of this resolution shall constitute a further material breach of Iraq's obligations and will be reported to the Council for assessment in accordance with paragraphs 11 and or 12 below."

- The US apparently interprets this as meaning a material breach occurs if the declaration submitted by Iraq contains any false statements. Other security council members interpret it as meaning the breach only occurs if Iraq also does not cooperate with the inspection process.

What's the Problem?

- The clause has the following logical structure:
  (A or B and C and D) entails E
  where
  - ✓ A = false statements [in the declarations submitted by Iraq]
  - ✓ B = omissions in the declarations submitted by Iraq
  - ✓ C = failure by Iraq at any time to comply with [this resolution]
  - ✓ D = [failure by Iraq at any time to] cooperate fully in the implementation of this resolution
  - ✓ E = a further material breach of Iraq's obligations
- So the two proposed readings are as follows:
  - ✓ ((A or (B and C and D)) entails E -- US interpretation
  - ✓ ((A or B) and (C and D)) entails E -- other security council members' interpretation

What is Conceptual Modeling?

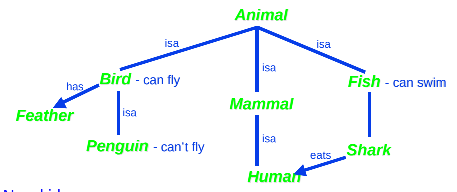


Conceptual Modeling is modeling using a standardized graphical notation

Origins

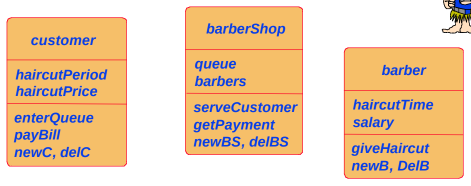
- There have been literally thousands of proposals for conceptual models, in several different areas within and outside Computer Science.
- Ross Quillian proposed in his PhD thesis ***semantic networks*** in order to model the structure of human memory (1966)
- Ole-Johan Dahl proposed in 1967 ***Simula***, an extension of the programming language ALGOL 60, for simulation applications which require some "world modeling"
- Jean-Robert Abrial proposed a ***semantic model*** in 1974, shortly followed by Peter Chen's ***Entity-relationship model*** (1975) as advances over logical data models, such as Codd's Relational model proposed only a few years before.
- Doug Ross proposed in the mid-70s the ***Structured Analysis and Design Technique*** (***SADT***) as a "language for communicating ideas". The technique was used by Softech, a Boston-based company, in order to specify requirements for software systems.

Semantic Networks



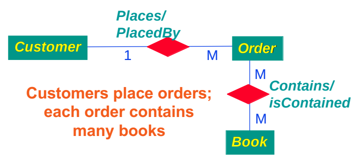
Novel ideas
- Models are built out of ***concepts*** and ***associations***
- ***Inheritance of attributes*** -- strict or default, single or multiple
- Computation defined in terms of ***spreading activation*** -- e.g., discovering the meaning of "horse food"
  - horse --> animal --> eat --> food
  - horse --> animal --> madeOf --> meat --> food

Simula (1966)



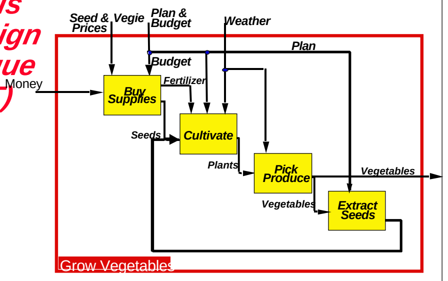
- Ole-Johan Dahl proposed it as an extension of the programming language ALGOL 60, for simulation applications.
- A (simulation) program consists of classes and instances; instances are partly data structures and partly processes.
- Instances ***model the simulated application***, classes define common features of instances, are organized into subclass hierarchies.

The Entity-Relationship Model



Novel ideas
- Assumes that application consists of ***entities*** and ***relationships*** (***ontological assumptions***)
- Shows how a conceptual schema can be mapped onto a logical one.
- [Abrial's semantic model was more akin to OO data models, but did offer entities and relations too]

Structured Analysis and Design Technique (SADT)



Novel Ideas
- Model operating environment of a software system.
- Application modeled in terms of ***data*** and ***activity.***
- Application models organized in terms of box-inside-box notation.

III. Use Cases

The Unified Modeling Language
Actors and Use Cases
How to Find Them

The Unified Modeling Language (UML)

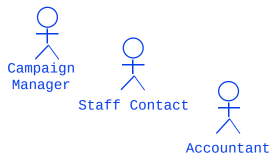
- Booch and Rumbaugh started working towards a unified modelling language (UML) in 1994 under the auspices of Rational Inc. They were later joined by Jacobson.
- UML only offers a notation, not a methodology for modeling (as various OOA techniques do).
- Combines Jacobson's use cases with Booch and Rumbaugh concepts for object modeling, along with statecharts.
- UML has been adopted by the Object Management Group {OMG) as an (object) modelling standard. OMG UML 1.0 is the first version of this new modelling standard.

Where Do We Start? Use Cases

- Use cases are descriptions of the functionality of the new system (or any artifact under design, for that matter!) from a user's perspective.
- They answer the question: How will the artifact be used, once it is built?
- Used to show the functions to be provided by the artifact, also which users will use which functions.
- Developed by Ivar Jacobson and friends [Jacobson92].

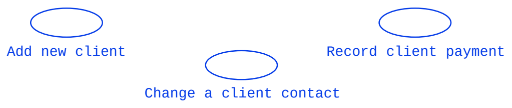
Actors

- An actor is anything that needs to exchange information with the artifact
- An actor could be a person, or another external, system.
- Actors define ***roles*** that users can play while using the artifact.



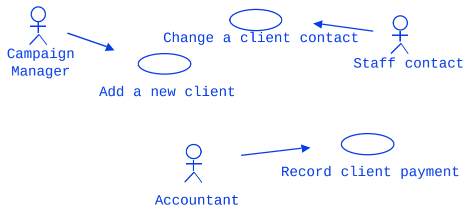
Use Cases

- A ***use case*** is a pattern of behavior which the new system is required to exhibit.
- Each use case is a sequence of related transactions performed by an actor and the system through a dialogue.
- To find use case, examine each actor and her needs, e.g.,
  - ✓Campaign Manager -- add a new client
  - ✓Staff Contact -- Change a client contact
  - ✓Accountant -- Record client payment

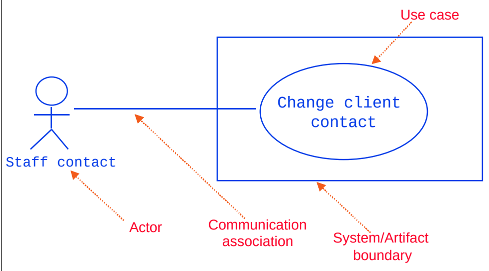


Use Case Diagrams

- Use case diagrams are created to capture the relationships between actors and use cases



Notation for Use Cases



Agate is an Advertising Company

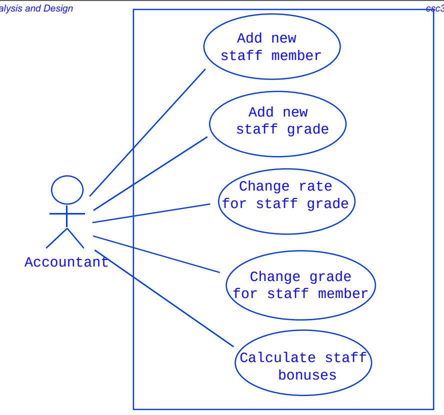
…which puts together advertising campaigns for client companies. Here is the breakdown of their staff:

Direction	Admin	Campaigns Mgt
1 Campaign	1 Office mgr	2 Campaign managers
1 Creative	3 Direction asst	3 Campaign marketers
1 Admin	4 Manager clerks	1 Editor in Chief
1 Finance	2 Receptionists	1 Creative Manager
	2 Clerks/typists	

Edition	Graphics
1 Filing clerk	6 Graphic designers
2 Editors	2 Photographers
4 Copy writers	

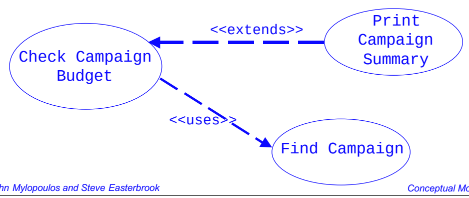
IT	Accounts Edition	Documentation
1 IT manager	1 Accountant manager	1 Media librarian
1 Network administrator	1 Credit controller	1 Resource libr
1 System admin	2 Accounts clerks	1 Knowledge worker
1 Analyst	2 Purchasing assistants	1 Computer tech

Agate Case Study



<<extends>> and <<uses>>

- <<extends>> implies that one use case adds behaviour to a base case; used to model a part of a use case that the user may see as optional system behavior; also models a separate sub-case which is executed conditionally.
- <<uses>>: adds behavior to a base case (like a procedure call); used to avoid describing the same flow of events several times, by putting the common behavior in a use case of its own.



Finding Actors

- Actors can be identified by answering the following questions:
  - ✓Who will be a primary user of the artifact? (primary actor)
  - ✓Who will need support from the artifact to do her daily tasks?
  - ✓Who will maintain, administrate, keep the artifact working? (secondary actor)
  - ✓Which hardware or other devices does the system need?
  - ✓With which other systems does the artifact need to interact with?
  - ✓Who or what has an interest in the results that the artifact produces ?
- Tip: don't consider only the users who directly use the artifact, but also others who need services from the artifact!

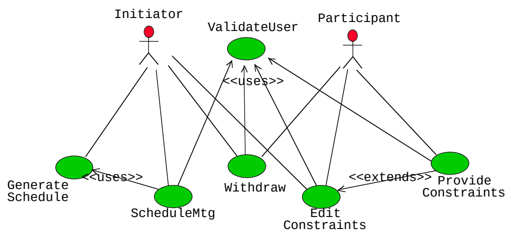
Finding Use Cases

For each actor, ask the following questions:
- Which functions does the actor require from the artifact? What does the actor need to do?
- Does the actor need to read, create, destroy, modify, or store some kinds of information in the artifact?
- Does the actor have to be notified about events in the artifact? Or, does the actor need to notify the artifact about something? What do those events require in terms of artifact functionality?
- Could the actor's daily work be simplified or made more efficient through new functions provided by the artifact?

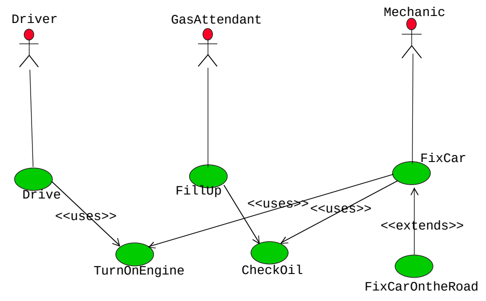
Documenting Use Cases

- For each use case, prepare a "flow of events" document, written from an actor's point of view.
- The document details what the system must provide to the actor when the use case is executed.
- Typical contents
  - ✓How the use case starts and ends;
  - ✓Normal flow of events;
  - ✓Alternate flow of events;
  - ✓Exceptional flow of events;

Use Cases for a Meeting Scheduling System



Use Cases for a Car



Additional Readings

- [Booch99] Booch, G. et al. *The Unified Modeling Language User Guide*, Chapters 2, 16, 17. Addison-Wesley, 1999.
- [Jacobson92] Jacobson, I. et all. *Object-Oriented Software Engineering: A Use-Case Driven Approach*, Addison-Wesley, 1992.
- [Schneider98] Schneider, G. et al. *Applying Use Cases*, Addison-Wesley, 1998.