

Lecture 2: Elaboration Tasks and Domain Modeling

Processes - Caveat

- Generic process applied to project with very different characteristics
- All kinds of traditional tasks (requirements, design, coding, testing) at all phases
- Confusing as to what is done in each abstract phase, such as Elaboration
- Major part of System Development: construct a process for the project

Elaboration Tasks

- Explore concepts from requirements/use cases
- Domain models: basic concepts and their inter-relationships
- Basic architectural concepts for system
- System sequence interactions – system events, UI basics

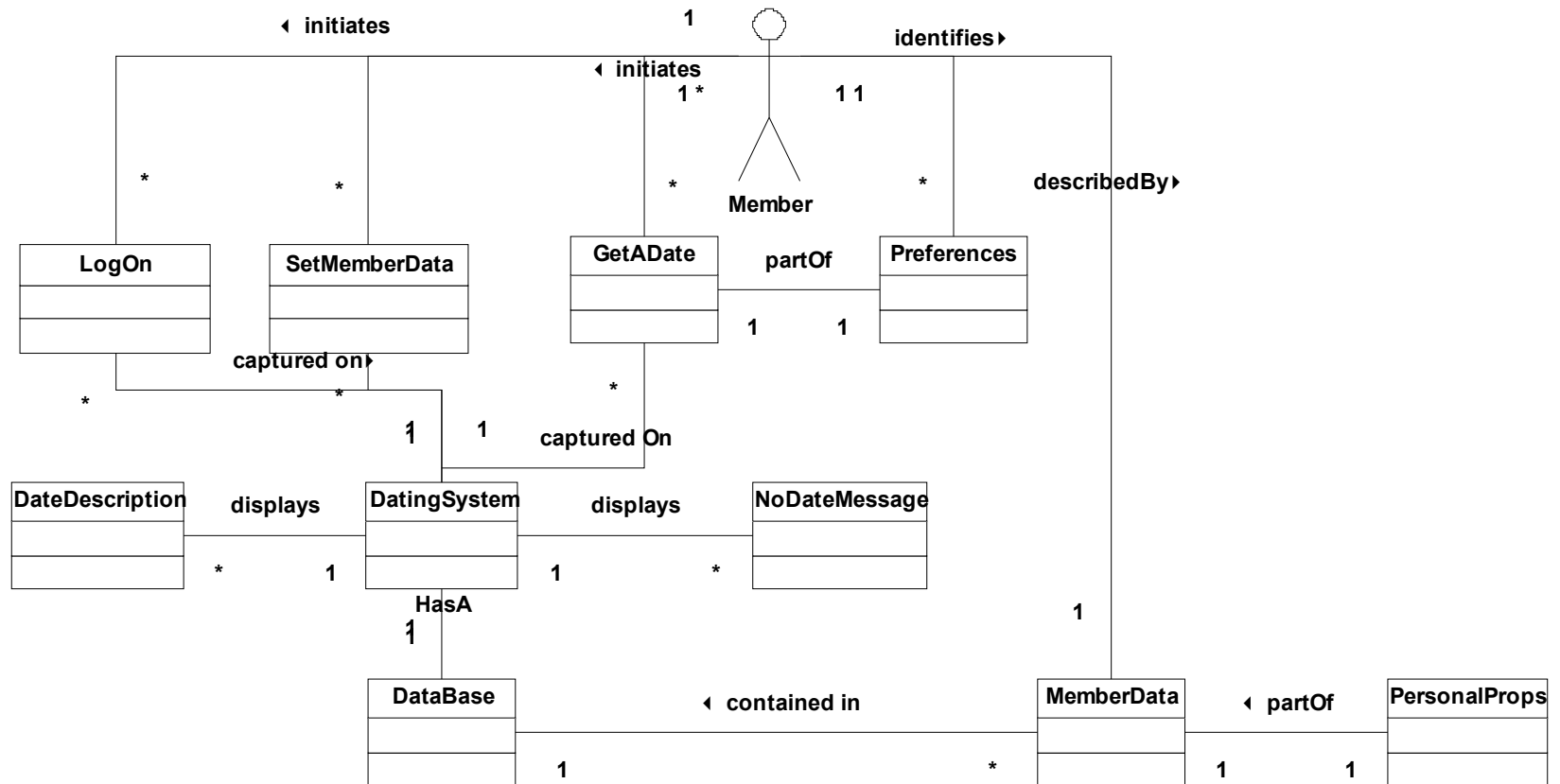
Stable System Design and Domain Models

- Systems change
 - Incremental development
 - Changing requirements
 - Post deployment enhancement
- Design for change
 - Simulation of problem domain
 - Functionality added to simulation

Domain Models

- Graphical Model
- Nodes:
 - Concepts/conceptual classes
 - Attributes: properties of classes
- Arcs: Relationships between concepts

Sample Partial Domain Model



Domain Model Does and Don'ts

- Static Model
 - E.g. Parts explosion diagram
- Real World Concepts
 - E.g. Gender Preference in Dating System Application
- Overdo concept list
- Dynamic Flow Chart
 - One entity sends a message to another
- Software Entities
 - E.g. GenderPreferenceTextField in DS
- Leave out concepts

Domain Models and Data Base Models

- Entity Relationship Diagrams
 - Similar to domain modes
- Used to identify:
 - Tables = Concepts
 - Columns/fields = Concept attributes
 - Table links = Concept relationships

Finding Concepts

- Define the boundary of the system
- Noun and noun phrase identification from prose descriptions and use cases
- Concept category checklists
- Special cases

Nouns and Noun Phrases for DS

- User
- Dating System
- LogOn
- Name
- Member
- Member Option
Choice
- Error Message
- GetADate
- DataBase
- SetMemberData
- Preferences
- DateDescription
- NoDateMessage
- MemberData
- PersonalProps

Concept Categories and the DS

- Physical Objects: user, member
- Descriptions: personal props, preferences
- Roles: member, dater, administrator
- Containers: DS Data Base
- Organizations: DS Accounting Dept.
- Events: logon, getADate, SetMemberData
- Abstract noun concepts: session

Descriptions

- Descriptions of other concepts that have a lifetime of their own
 - E.g. dater properties in Dating System
 - May be stored in data base
 - May be changed
- Alternative is to make the information an attribute of a concept
 - primitive class attributes vs class variables

Events

- Things to which system will have to respond
 - E.g. logon, date-request
- Object oriented approach to events: create an instance of an event object which “knows how to process such events”
 - E.g. Java GUI: create event object for stack that knows source object for event, which will have a listener list of objects with method for processing event

Associations

- Describe semantically meaningful relationships between concepts
- Give the structure of the domain model
- May be denoted with roles and multiplicity designators
- Important not to miss concepts
- Important not to over do associations

Finding Associations

- Need to know criteria
 - Needed for operation of system
 - Have a lifetime of importance
 - E.g. link between DS member and Account is necessary.
 - E.g. link between administrator and GetaDate request is not necessary.
- Association category checklists

Association Category Checklists

- A is a physical or logical part of B
 - E.g. Physical Characteristics and Personal Characteristics are part of Dater Properties
- A is physically or logically contained in B
 - E.g. MembershipData contained in MemberDB
- A is recorded in/by B
 - DatingSystem records occurrence of User Asking for a Date event

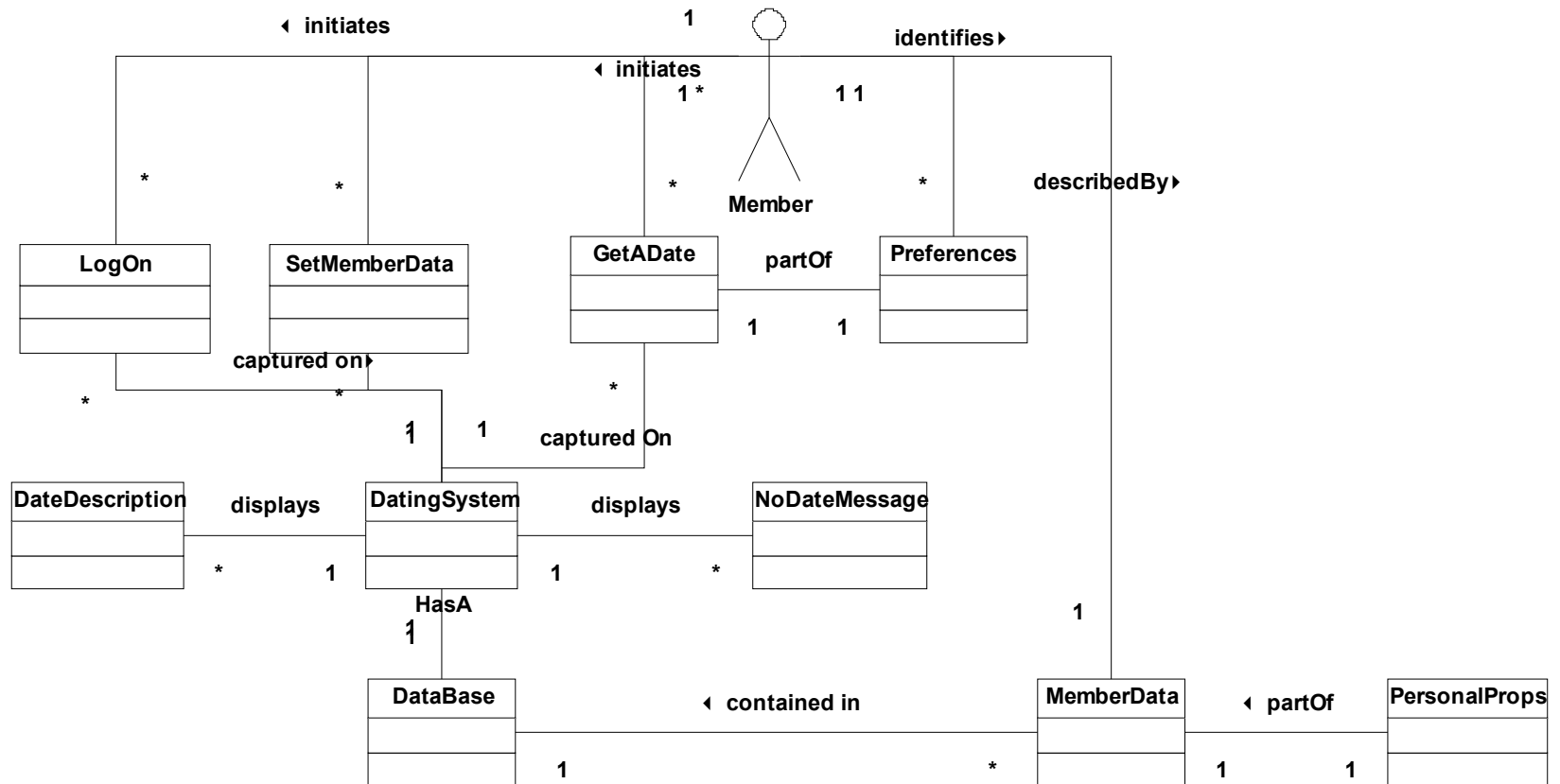
Associations and Class Design

- Logical Parts
- Class Variables: the objects in one class consist of an aggregation of objects from other classes
- Contained in
- A vector or other container class contains objects of one or more other classes

Association Details

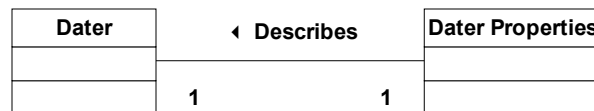
- Names
 - E.g. Dater Properties-(describe)-Dater
- Multiplicity
 - One to one, many to many, one to many
 - E.g. Dater Properties-*- (describe)-1 -Dater
- Direction of association

Domain Model Example



Attributes

- Problem domain properties
- Should have primitive data type values
- If property is a (complex) object, should be a separate concept with a linking association
 - E.g. Dater has Dater Properties. This is a complex object and should be its own concept. Linking association:



Associations vs Attributes – When?

- Has Parts to it
- Special operations associated with it
- Has its own attributes
- Quantity with a unit
- Abstraction of an entity with above properties
- Standard data types (integer, string, ...)
- Quantity with a unit
- Note: use of foreign keys (primitive key value used to identify a complex object) not acceptable

Associations vs Attributes - Why?

- Allows postponing of details
- Leads to a design that facilitates changes
 - E.g. DaterProperties
 - Suppose we make these attributes of Member, whose instances are daters, and for which there will be a record in the DB?
 - Instead: use an association with a separate class

Domain Models and Attributes

- Not enough room in graphical models
- Give separate documents showing domain model concepts with their attributes
- E.g.

Logon
-Name: : string
-UserType : char

SetMemberData
-Gender : bool
-Religion : string
-Occupation : string

UML and Domain Models

- UML describes diagram types
- Class diagram
 - Conceptual classes as in Domain Model
 - Attributes but no methods
 - Design classes from design activities
 - Graphical representation of PL classes

Domain Models and System Increments

- Construct for current iteration concepts, associations and attributes
- Will augment with later increments
- May include concepts not in current increment as part of “extra is better than missing” approach