1 Domain Modeling

Domain modeling is a method for identifying the relative concepts and tasks of a domain. It is used to fix the terminology and the fundamental activities of the domain.

Domain Model

- Goal:
  - Decompose domain into concepts or objects
  - Represent the real word (as defined by requirements specifications)
- Creation:
  - Identify a set of conceptual classes and fundamental actions
  - Completed iteratively, forms basis or software design
- Synonyms:
  - Conceptual Model, domain object model, analysis object model

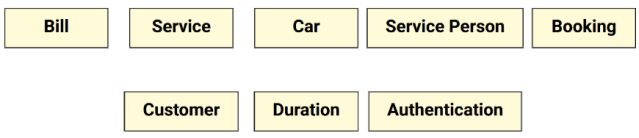
Conceptual Classes

- Represents ideas, things or objects in the domain
- Attributes:
  - Name or Symbol representing the class
  - Intention
  - Extension (contains domain elements)

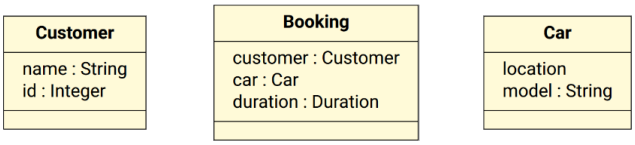
1.1 Visualization of Domain Models

Domain Models are visualized using **UML Class Diagrams** with suitable restrictions to emphasize domain modeling:

- Only domain objects and conceptual classes
- Only associations, no aggregation, no composition
- Classes may have attributes but no operations



Example of Conceptional class (Car sharing)



Example of conceptual classes with attributes (domain objects)

Hereby an objext is defined as an individual thing with a state and relations to other objects.

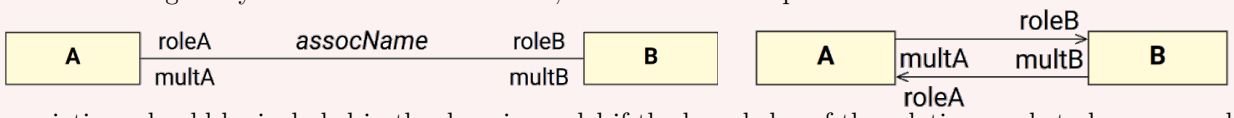
UML Class Elements and Conventions

- Class Name: Always starts with an upper case letter
- Attributes:
  - Name: starts with a lower case letter
  - Type: Pre-defined type or other domain model class (can be omitted)
- Derived Attribute:
  - Name: prefixed by a slash, followed by a lower case letter
  - Describes a value computable from existing information

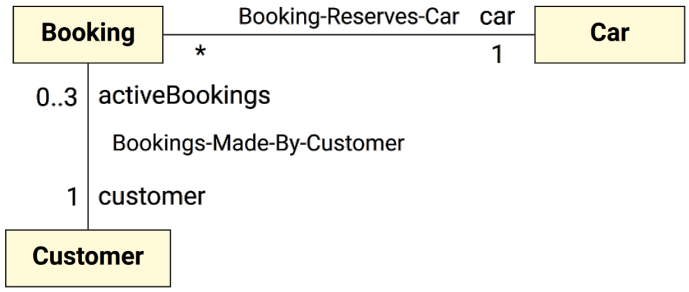
UML Associations / Relations

An association is a relation among classes. It consists of the following:

- Name: (optional) Should be done according to the **Class name-verb phrase-Class name** format:
  - Player-Stands-on-Square
  - Sale-Paid-by-CashPayment
  - Customer-Traveled-by-Vehicle
- Two Roles (associations):
  - Name: Defaults to class name in lowercase
  - Multiplicity: Defaults to 1.
    Possible: * (arbitrary/all) and a..b (Range, upper bound inclusive)
  - Navigability: Defaults to bidirectional, not used for conceptual classe.



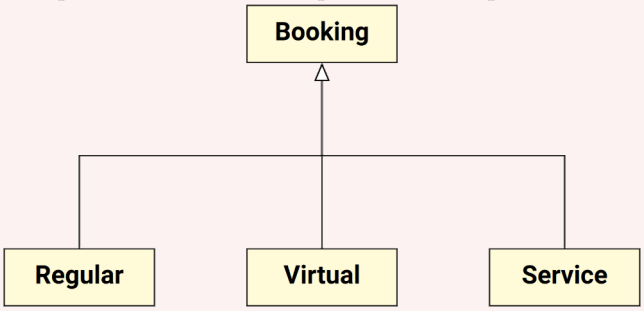
Associations should be included in the domain model if the knowledge of the relation needs to be preserved. For example: The relation between a bill and its entries needs to be preserved. However the relation between a user and their recent searches is not necessarily important.



Example of UML Class Association

Class Generalizations

Multiple classes can be comprised of multiple subclasses:

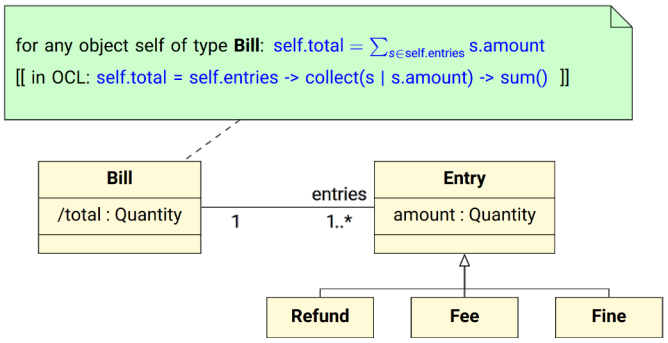


1.2 Eliciation of Domain Models

Workflow

1. Find the conceptual classes
   Strategies:
   - Re-Use or modify existing model
   - Use a category list
   - Identify noun phrases
2. Draw elicited concepts as classes in a UML class diagram
3. Add attributes
4. Add associations

1.2.1 Re-Using Existing Models



1.2.2 List of Conceptual Classes Categories

Some Categories:

- Physical objects
- Specifications of things
- Locations
- Events
- Transactions
- etc.

Conceptual Class Category	Conceptual Classes (in CaSh)
Business transaction	Bill, Payment
Transaction line item	Entry
Product or service related to a transaction or to a transaction line item	Refund, Rent, Fine
Place where transaction is recorded	Registry
Roles of people or organizations related to a transaction (actors in use cases)	Customer, Accountant
Location where transaction executed	Website
Noteworthy events, with a time or place that needs to be remembered	Bill, Booking

1.2.3 Noun Identification

Workflow:

1. Identify nouns and noun phrases in textual description (Use Cases for example) of domain
2. Consider them as a candidate for a conceptual class or attribute

Criteria for inclusion of conceptual classes:

- Must carry information not available/computable from other sources
- Must have specific semantic in relation to the business

Can only be partially automated due to the ambiguity of natural language

1.3 Description Classes

A description class contains information that describes an entity. For example, a description class for a car would contain information about the car's make, model, color, etc.

A description class should be added to the model if:

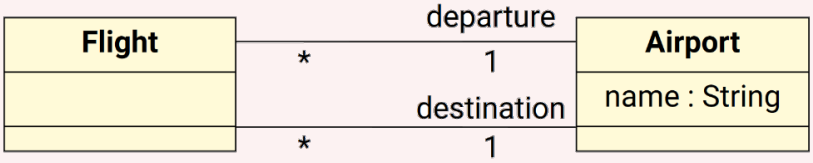
- Information about the entity is required, regardless of whether an instance of the entity even exists.
- Deleting an instance of the entity would result in loss of information.
- Redundant or duplicate information is reduced

Class or Attribute

When deciding if a piece of information should be included as an attribute or a class: If notion C is not considered:

- Number
- Text
- Date

that usually indicates that it should be a **class**.



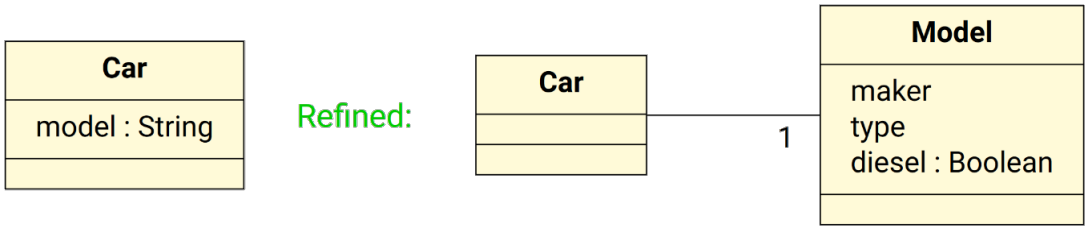
When considering the destination of a flight, it makes more sense to put that information into a class Airport, instead of making it an attribute of flight. Also allows for better alotment of additional info.

When to use Attributes or Associations

- Attributes should always describe primitive datatypes:
  - Boolean, Integer, Character, String…
  - Dates, Address, Colors, Phone Numbers…
- Quantities may be modelled as classes to attach units:
  - currency: EUR, USD, CAD…
  - distance: meters, miles, millimeters…
- Relations between conceptual classes are always associations

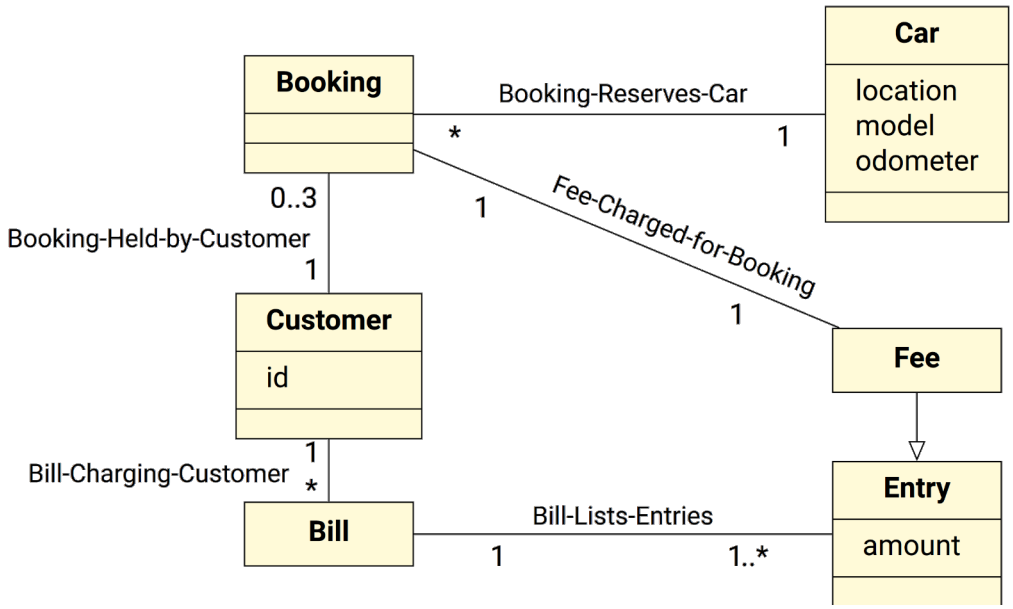
1.3.1 Refining the model

Initially it is very convenient to type attributes with Strings. It is a generic type that avoids premature decision. Later on it can be refined into a description class:



Obviously a string can only be refined if it actually contains more information that can be shown differently.

In general, the domain model serves as an inspiration for the design model later on.



An example Domain Model

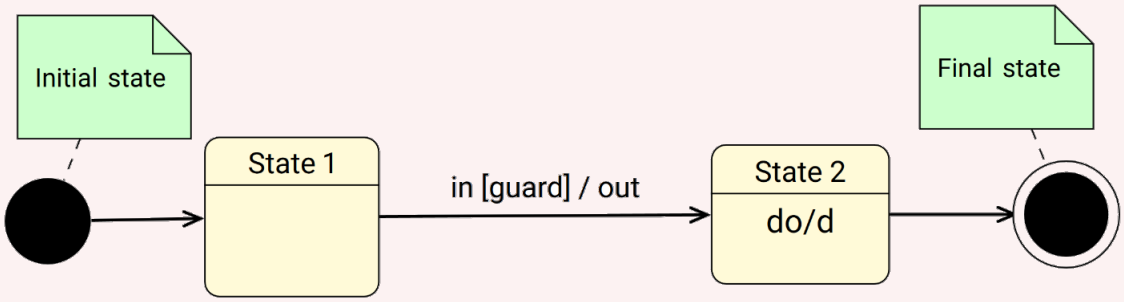
1.4 Behavioural Domain Modelling

Class Diagrams only model static aspects such as classes, objects, attributes, associations and functions.

What we are missing are the behaviours, the sequence of actions and how they change the state of the system and under which condition.

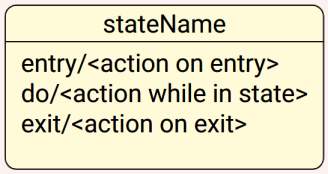
These behaviours can be displayed as UML State Machines Diagrams.

UML State Machine Diagrams



This can be read as: *When in State 1, if input in is observed and guard is true, then output out happens and current state becomes state 2. In state 2 perform the (interruptible) action d.*
So in General UML State Machine Diagrams show States and their conditions for transitioning as in- and outputs and guards.

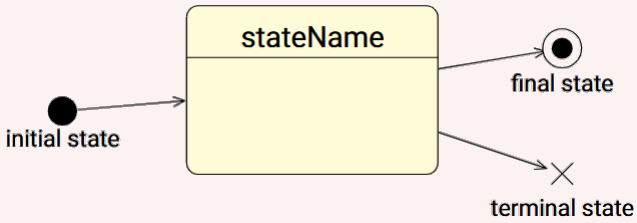
Basic States



States consist of the following:

- Name: Short description of what the state represents
- Actions: Executed Operations
  - Entry: Action performed on state entry
  - Do: Action performed while in state (until terminated or state is left)
  - Exit: Action performed on state exit

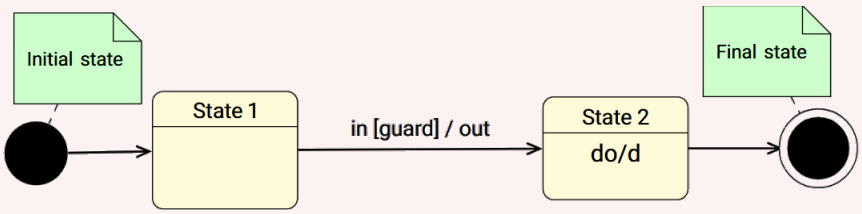
Special States



There are some special states:

- Initial State: Has a single transition to first entered state
  May be labeled by object creation event
- Final State: Indicates completion of scenario
- Terminal State: Completion and executing object destroyed

Transitions

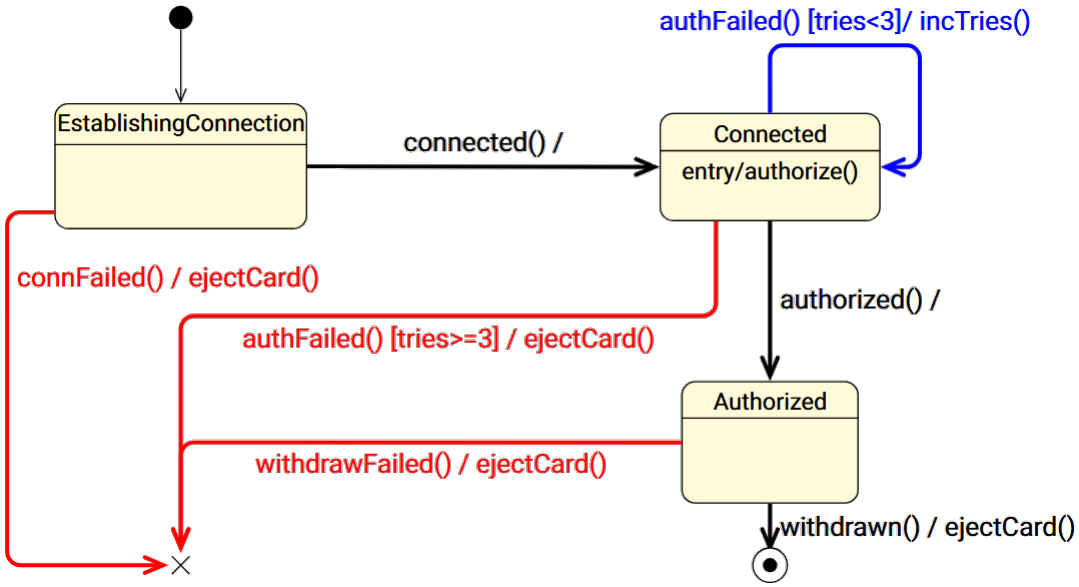


A transition label is usually in the format **input?** [**guard**]? / **output?**. Hereby all components are optional (therfore ?).

- Input (Trigger) events are observations:
  - call event (start of operation)
  - time event (e.g. time spent in a state)
  - change event (value of attribute has changed)
- Guard is a boolean expression (for example: if input == value)
- Output (action) is an operation

In general, the purpose of state machine diagrams are:

- Capturing action sequences of a use case
- Combine related use cases
- Clarify the states of an object
- Clarify protocols
- Validate the domain model
- Complete the domain model (properties, actions)
- Model **only** non-trivial behaviour



An example State Machine Diagram