# Embedded Software: Model Based Design (MBD)

Franco Fummi

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Version 1.1

# Contents

- Introduction
  - Motivations for Model Based Design
  - Existing MBD approaches
- UML
  - Main characteristics
  - UML for Embedded software
- Case study
  - Coffee vending machine
- References

# INTRODUCTION

# Embedded SW: what?

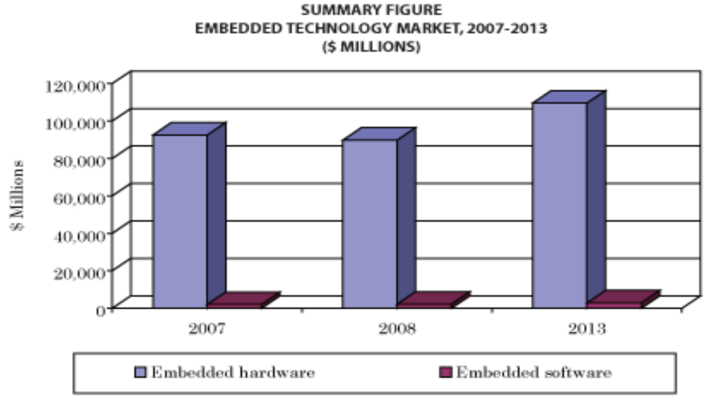
- It is a specific-purpose software
  - tightly integrated with the underlying execution platform
  - that constantly reacts to events, and
  - mixes control and data flows
- Its main role is not transformation of data, but interaction with the physical world
  - It executes on machine that are not only computers …
    - … they are cars, planes, phones, medical equipments, toys, manufacturing systems, …

# ESW: characteristics

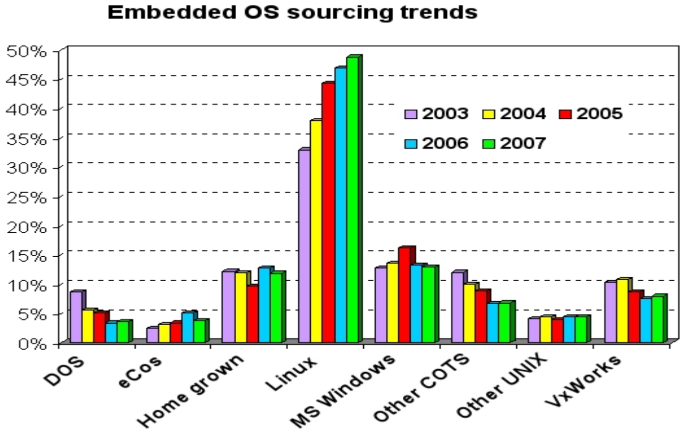
- Timeliness
  - Physical processes evolve over time
  - A “late computation” is not just in delay, it is incorrect!
- Concurrency
  - Signals from environment can arrive simultaneously
  - Disjoint but parallel activities may be monitored
- Liveness
  - In the Turing view of computation, all nonterminating programs are defective programs
  - In embedded computing, terminating programs are defective!
- Reactivity
  - ESW are real-time constrained and safety critical systems that react continuously to their environment
- Heterogeneity
  - ESW mixes computational styles and implementation technologies

# ES market: trend

- Expected to increase from $92.0 billion in 2008 to $112.5 billion by the end of 2013:
  - a compound annual growth rate (CAGR) of 4.1%
  - embedded hardware from $89.8 billion in 2008 to $109.6 billion in 2013
  - embedded software from $2.2 billion in 2008 to $2.9 billion in 2013, for a CAGR of 5.6%.




Source: http://www.bccresearch.com/report/IFT016C.html




Source: http://www.linuxfordevices.com

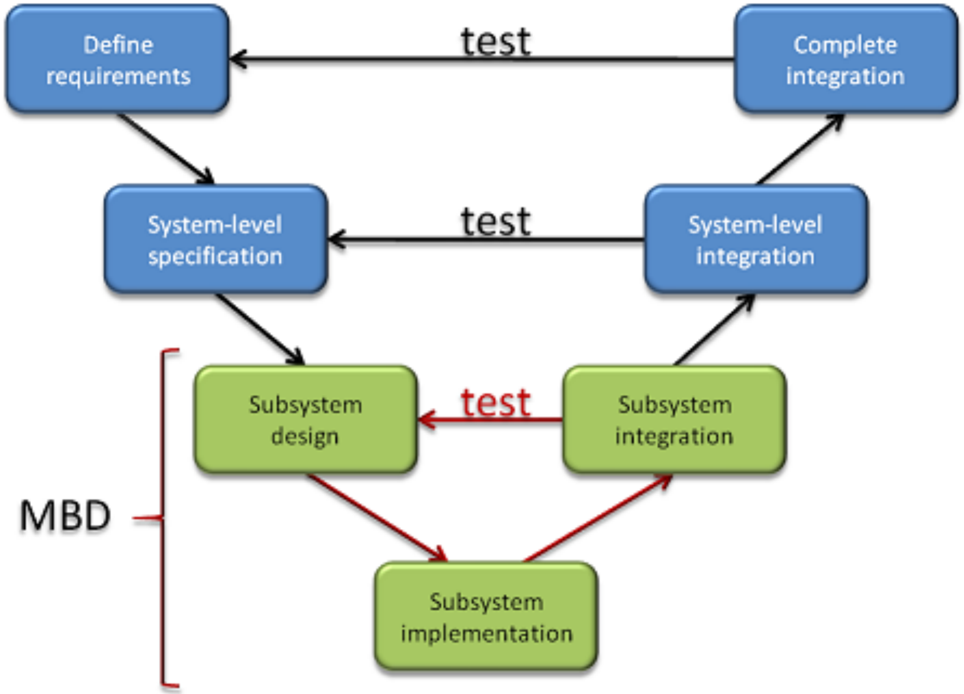
# ESW: role

- Previous data highlights that:
  - ESW is the component of an Embedded System that is making and it will even more make the difference
    - ESW has the central role in the value of an ES
- ESW must thus be:
  - rapidly developed
  - rapidly modifiable and extendable
  - compliant to customer specifications
  - easily portable among ES platforms

# ESW: design constraints

- ESW design implies conflicting design constraints:
  - efficient, effective, low computational cost, low memory…
    - thigh integration with the ES platform
  - reusable, rapidly developed, maintainable, portable…
    - abstract and independent from the ES platform
- ESW design solution solving the conflict:
  - design based on abstract models
  - automatic code generations from the abstract models
    - in few words: Model Based Design (MBD)

# MBD: general flow



# MBD: solutions on the market (I)

- Matlab
  - application field:
    - Algorithms development
    - Rapid prototyping of HW/SW designs
    - Performance analysis
    - Optimization analysis
  - disadvantages:
    - Supports only functional specification

# MBD: solutions on the market (II)

- Modelica
  - application field:
    - Technical systems design, e.g.,
      mechanical, electrical, thermal, hydraulic, pneumatic, fluid, control systems
    - Dynamic behavior specification described by differential, algebraic, and discrete equations
    - Performance analysis
  - disadvantages:
    - Supports mainly functional/behavioral specification

# MBD: solutions on the market (III)

- Simulink
  - application field:
    - Dynamical and control systems design
    - System behaviors described by using a library of graphical blocks or differential, algebraic, and discrete equations
    - Performance analysis
    - Optimization analysis
  - disadvantages:
    - Supports mainly functional/behavioral specification
    - Partially supports the structural partitioning of the system

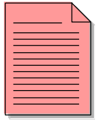
# MBD: solutions on the market (IV)

- UML-based flows
  - Structural specification
  - Functional specification
  - Use case specification
- Why UML is the most suited MBD for ESW design
  - Graphical specification are easy to be interpreted
  - Specification at different abstraction levels
    - Requirements level (use cases)
    - System level (structural view)
    - Integration level (subsystems interaction view)
    - Unit level (subsystem functional view)

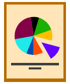
# UNIFIED MODELING LANGUAGE (UML)

# UML Diagram – What is UML?

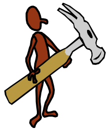
- The Unified Modeling Language (UML) is a standard  language for



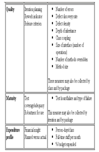
Specifying



Visualizing



Constructing

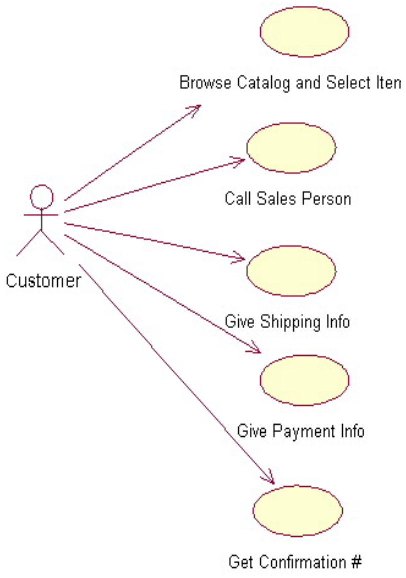


Documenting

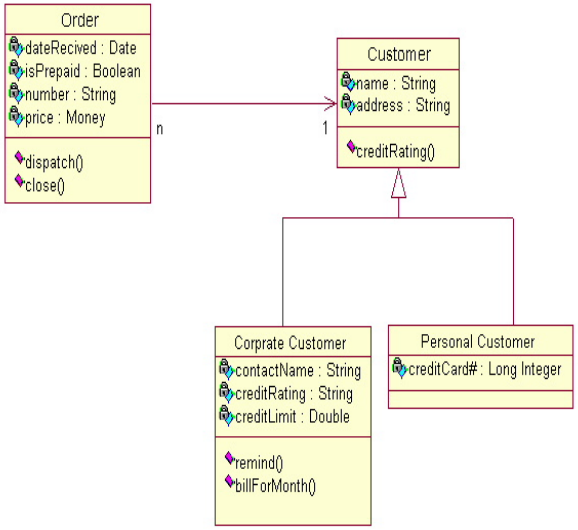
Business Modeling

Communications

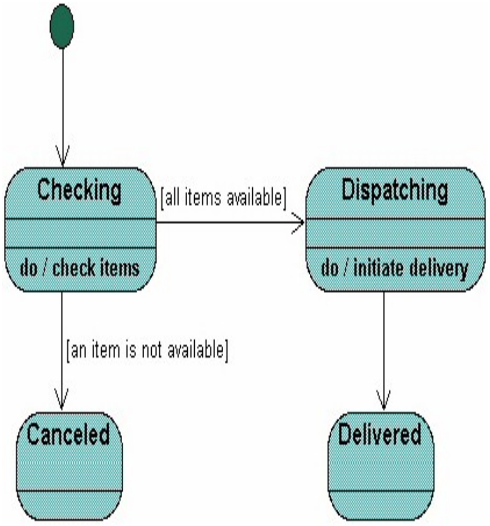
# Different Views




Users




Designers



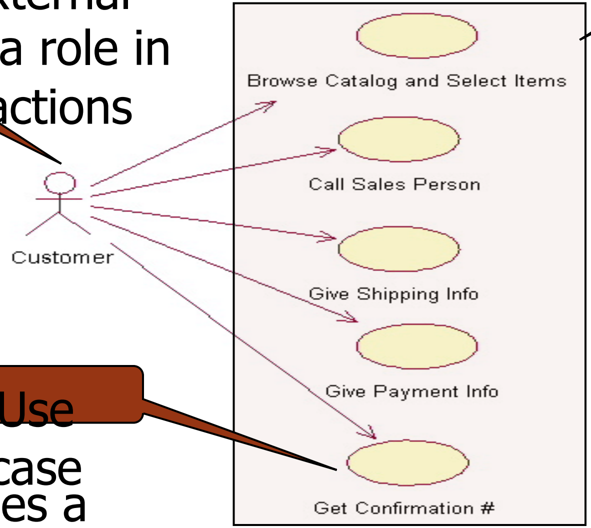

Analyzers

# Use Case Diagram

**Actor:**

An actor is a person, organization, or external system that plays a role in one or more interactions with your system

Actor




**System boundary**

• Overview the usage requirements
• Presentations project stakeholders
• "The meat" of the actual requirements

**System boundary:**

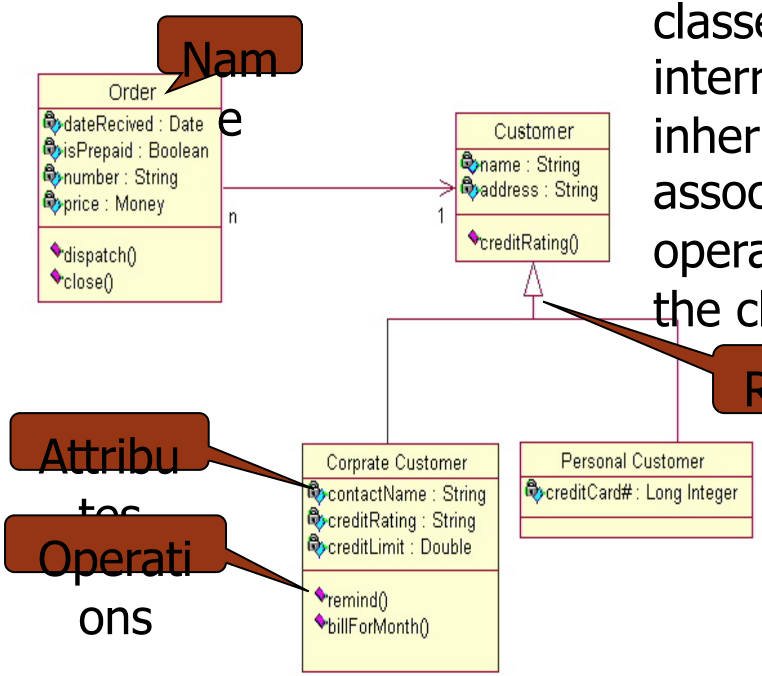
Indicates the scope of your system. Anything within the box represents functionality that is in scope and anything outside the box is not

**Use case:**

Use case

A use case describes a sequence of actions that provide something of measurable value to an

# Class Diagram

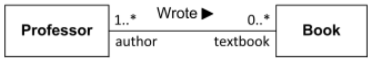



Class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), and the operations and attributes of the classes

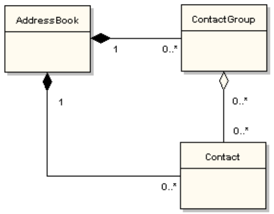
- Associations
- Aggregation
- Generalization

# Relationships between Class Diagrams

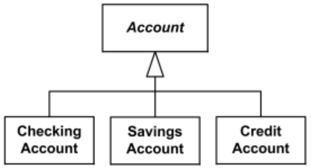
- Association -- relationship between instances of two classes
  - There is an association between two classes if an instance of one class must know about the other in order to perform its work



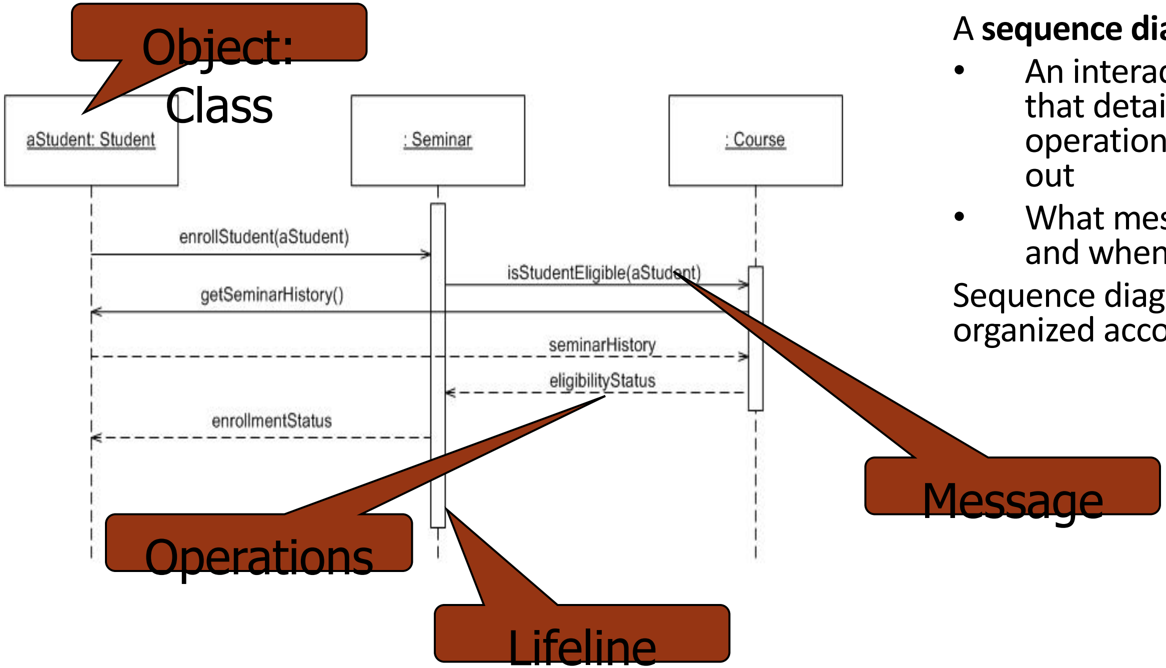

- Aggregation -- an association in which one class belongs to a collection
  - An aggregation has a diamond end pointing to the part containing the whole




- Generalization -- an inheritance link indicating one class is a superclass of the other
  - A generalization has a triangle pointing to the superclass



# Sequence Diagram

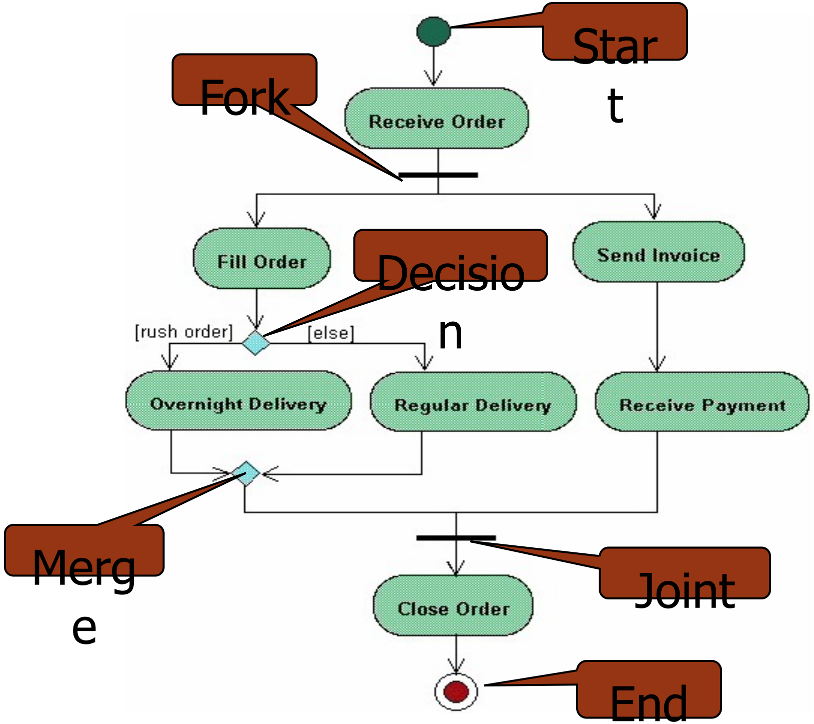



A **sequence diagram** is

- An interaction diagram that details how operations are carried out
- What messages are sent and when

Sequence diagrams are organized according to time

# Activities Diagram




Activity diagrams describe the workflow behaviour of a system

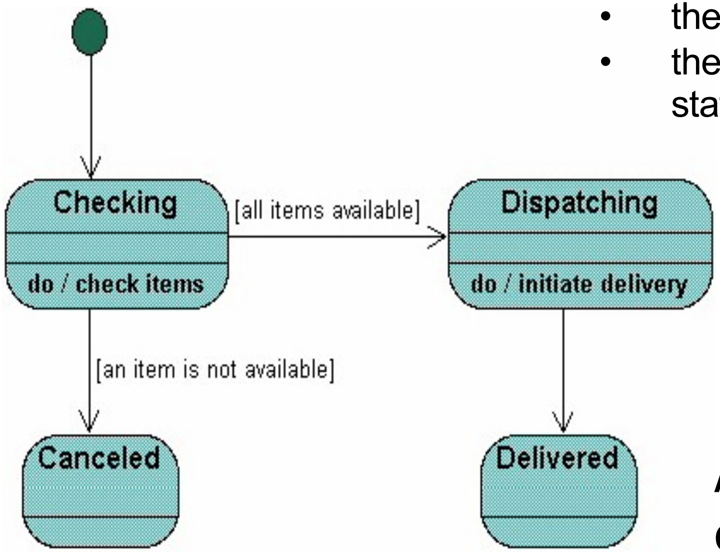
**Fork** denotes the beginning of parallel activity

**Joint** denotes the end of parallel processing. All flows going into the join must reach it before processing may continue

**Decision** (a diamond with one flow entering and several leaving): the flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.

**Merge** (a diamond with several flows entering and one leaving): the implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow

# State Machine Diagram




A State Machine diagram shows:

- the possible states of the object and
- the transitions that cause a change in state

?

What is different between activities and State Machine diagram

Activity diagram is a flow chart which shows the flow of activity of a process

State diagram shows the object undergoing a process

# UML for Embedded Software

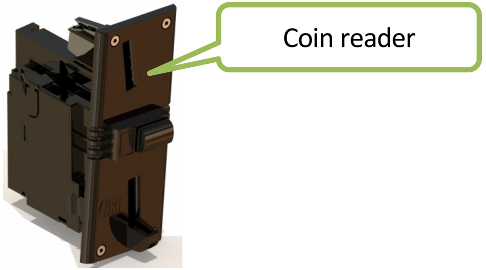
- Which are the diagrams useful for ESW design?
  - Use case
    - only for a first roundtable with the customer… too abstract
  - Class
    - not very useful since too high-level for ESW data structures
  - Sequence
    - they capture timing relationships potentially interesting for ESW even if it is hard to automatically coding
  - Activities
    - too far away from actual code
  - State machine
    - perfect trade-off among abstraction and closeness to actual code
      - they allow to make an abstract representation of ESW, but maintaining the contact to the generated code

Coffee vending machine

# CASE STUDY

# Specifications

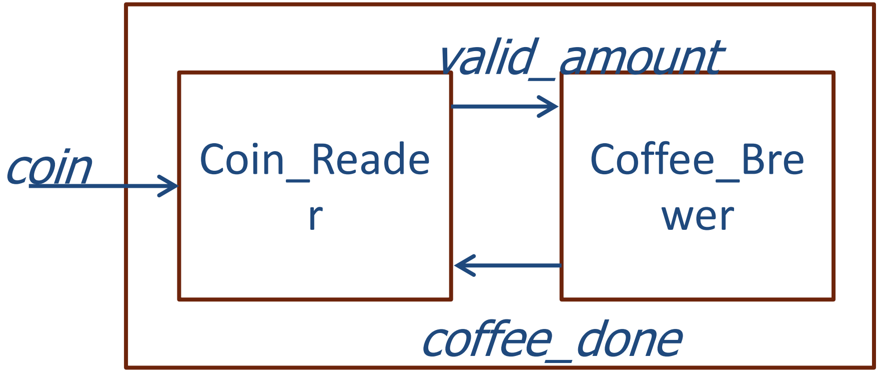
- The machine takes 5, 10, 25-cent coins
- The coffee cost is 35 cents




# Requirements

- The maximum amount of money inserted is 55 cents
- When the amount of money is sufficient the machine starts preparing the coffee
- If the amount of money is sufficient for preparing a coffee, the coin slot is closed
- If the serving of a coffee is done, then the coin slot has to be opened for accepting new money
- The conclusion of the coffee preparation is combined with a buzzer notification
- The machine is cleaned before the coffee brewing starts

# Module design



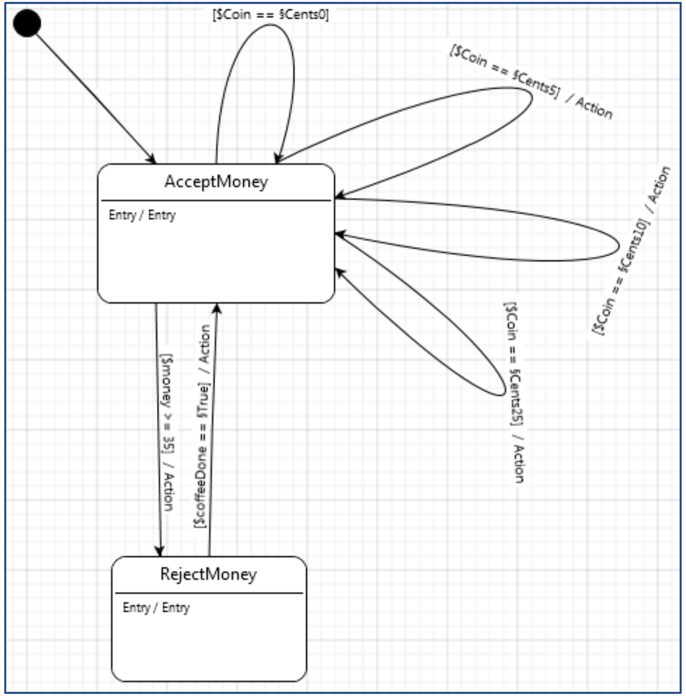

Coin_Reader

├ INPUT PORT: *coin*
├ INPUT VAR: *cofee_done*
├ OUTPUT VAR: *valid_amount*
└ LOCAL: *money*

Cofeee Brewer

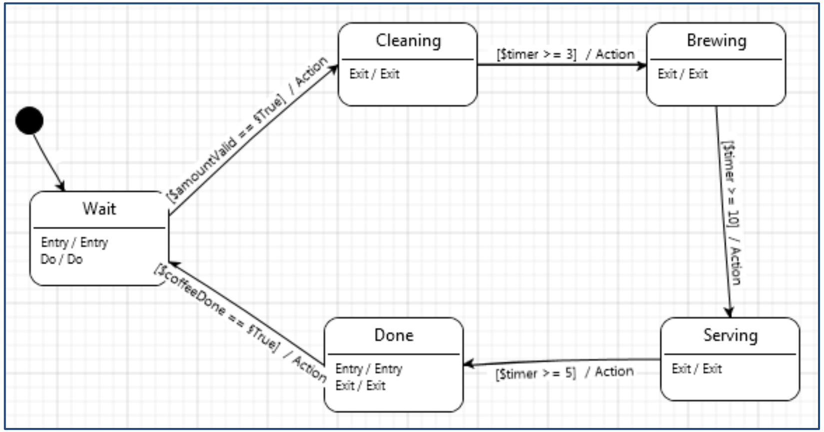
├ INPUT PORT:
├ INPUT VAR: *valid_amount*
├ OUTPUT VAR: *coffee_done*
└ LOCAL: *timer*

# From specifications to model




Coin_Reader

Coffee_Brewer



# References

- http://www.omg.org/news/whitepapers/#UML
- http://www.sparxsystems.com/resources/uml2_tutorial/