1 Introduction

1.1 The Software is level of abstraction

The history of software development is a history of raising the level of abstraction.

1.2 What is Executable UML

1.2.1 Executable UML a sub-set of UML. Why?

Allows us to focus on 'modelling' rather than the intricate and often confusing detail of UML.

1.2.2 UML and Modelling

- UML is just a notation for representing models.
- Executable UML is the higher layer of abstraction of software development. It is abstraction both on Specific **programming languages** and **decisions** about the organization.
- An executable UML specification comprises a set of **models** represented as **diagrams** that describe and define the conceptualization and behavior of the requirements.

1.2.3 Terminology defination in UML

- State Machine: the **objects** (the instances of the classes) may have **lifecycles** (behaviors over time) that are abstracted as **state machines**.
- Event: The behaviour of the system is driven by objects moving from one stage in their lifecycles to another in response to events.
- Procedure/Action: Each **state machine** has a set of **procedures***,* one of which is executed when the object changes state, thus establishing the new state. Each **procedure** comprises a set of **actions**.
- Models: These three models—the **class model, the state machines for the classes, and the states' procedures**—form a complete definition of the subject matter under study.

1.3 Model compliers

- The executable UML abstracted away the organization of hardware and software.
- An executable UML *model compiler* turns an executable UML model into an implementation using a set of decisions about the target hardware and software environment.
- An executable UML model compiler make decisions about a particular hardware and software environment.
- The performance of the model compiler may depend on the allocations of application model elements.

1.4 Domain models are executable

- Dynamic analysis can significantly improve domain understanding and agreement among stakeholders.
- Allows testing to start very early in the software development life-cycle Where the cost of fixing problems is much cheaper.
- Facilitates improved translation of models into artifacts such as code and documentation

This is why executable UML is so powerful—by separating the model of the subject matter from its software structure, the two aspects can be changed independently, making it easier to modify one without adversely affecting the other.

2 Domain

The benefit of modelling process:

- **Enhance stakeholder engagement**
- **Improve understanding**
- **Capture intellectual effort in a reusable form**
- **Increase productivity during system construction**

2.1 The System Model

Each domain is an autonomous world inhabited by conceptual entities.
Bridge: Domain dependency is a different beast from compilation dependency between packages, so we give it a different name, a *bridge*.
Use cases help establish the domains for which we build executable UML models. **Use cases** also provide us with the vocabulary we need to be able to build appropriate abstractions in each domain.

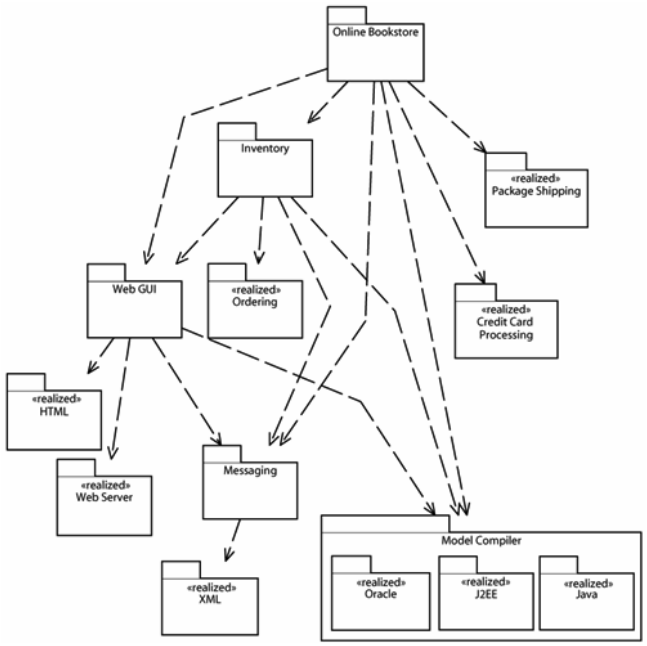
2.2 The domains

2.2.1 Why use independent domains?

Maintain clear separation of concerns.
Each domain deals with a specific 'subject mater'.
Domains are autonomous.
Domain models are complete and detailed.
Domain models are integrated to form system specification later.
The domain chart provides the highest-level view of the system.

2.2.2 The examples of domain chart and domain

Figure 2.1. Domain Chart



Some examples of domains:

- Application Domains
  – Flight Control
  – Insurance Claims
  – Course administration
- Service Domains
  – Data storage, backup/recovery ...
  – Security
  – Messaging
- Interface Domains
  – Web GUI
  – UI Look and Feel
- Architecture Domains
  – J2EE
  – LAMP (Linux, Apache, MySql, PHP)
- Implementation Domains
  – Java
  – HTML

Domains deal with autonomous subject matter. They therefore separate concerns.

2.3 The steps of design domain

- **Gathering** and **understanding** requirements

- Select or invent the appropriate **abstraction** and make detailed decisions about how the domain works.
- developer must understand the domain's mission and have sufficient detailed information to make decisions about the exact behavior of the domain.

Models are both abstract and detailed at the same time.

The executable UML model for each domain comprises a set of tightly connected class, state, and action models.

2.4 Domain will be modelled in term of Data, State and Process

- Data
  – Entities
  – Relationships
- State
  – The known states of entities involved in a system
  – The events which cause transitions between states
- Process
  – The transformation of data during state transitions
  – Communication between parts of a system

2.5 Two approaches to model the complete system:

- Mellor & Balcer
  – 'Bridges' between domains
  – Similar to (and pre-dates) aspect-oriented programming concepts
- Flint
  – Sets of requirements stated in terms of knowledge captured in reusable domain models
  – 'Archetypes' for various classes of system

3 Data Modelling

Modeling a Single Domain with **Classes and Associations:**
– Real-world entities abstracted as Classes
– Real-world relationships abstracted as Associations

3.1 Classes:

A class is an abstraction of a set of conceptual entities in a domain which have common characteristics.
They formalize the characteristics and behavior of real-world things.
Classes can be:
Tangible Thing,
Roles,
Incidents,
Interactions,

- Purchase, library loan

Specifications

- Assessment scheme

Instances of class:
The objects that contain specific data in it.
Attributes of class:
Each attribute represents a single piece of data which every instance of that class must have
Each instance may have a different value for the same attribute
Each attribute must have a type
It is important to note that ...
Novice modellers often focus on real-world things rather than the relationships between them
However, the interesting and difficult aspects of a system are largely determined by the relationships
– This is where you must focus your attention

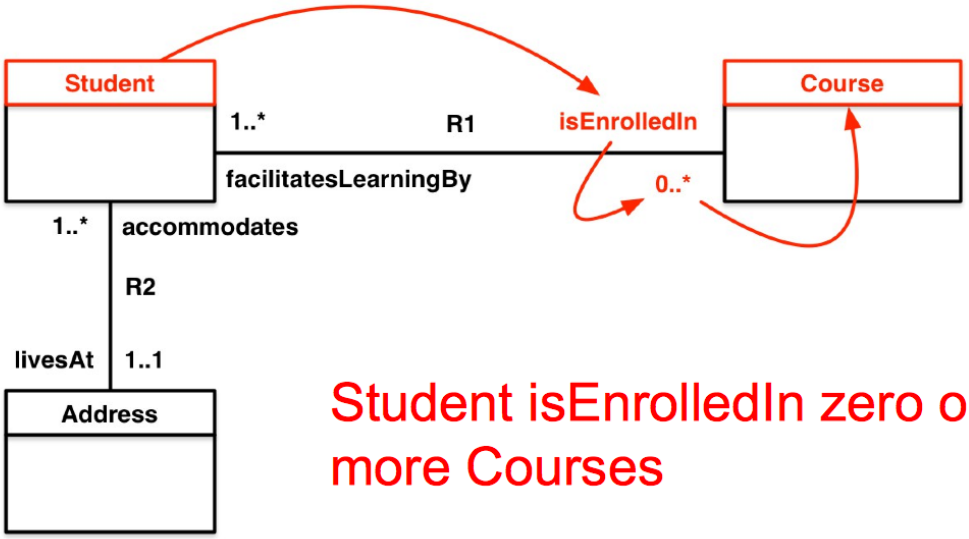
3.2 Associations

Associations are abstraction of real-world relationships between real-world things.
They formalize how real-world things relate to each other.

Characteristics:
Each association has a name, such as R01, R02, … , R56.
Association meaning: a string at each end of the association.
Multiplicities are used to specify the number of class instances that participate in a association.
Unconditional:
– 1..1 – exactly one instance
– 1..* - one or more instances
Conditional:
– 0..1 – zero or one instance
– 0..* - zero or many instances

Reading Associations:

Reading associations

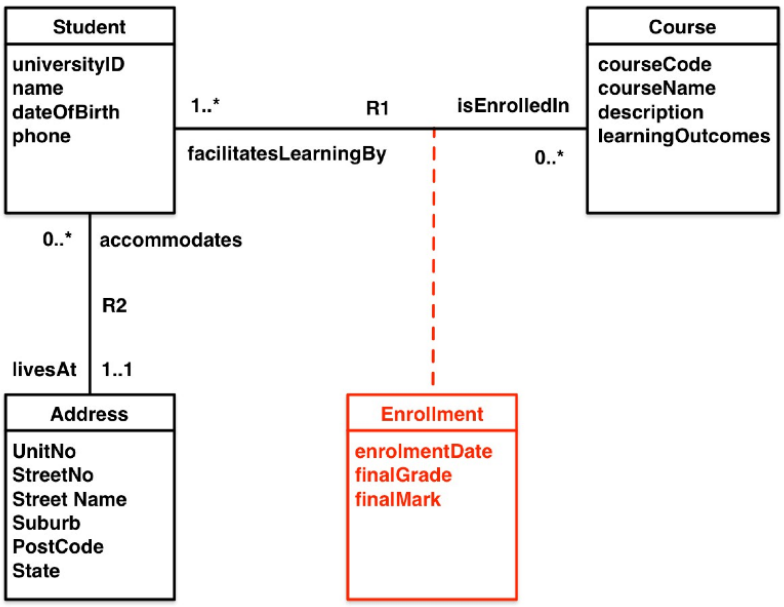


Student isEnrolledIn zero or more Courses

3.3 Associative Classes:

In some cases, associations are more complex
– They may have their own attributes
– They may participate in other relationships
– They may have their own behaviour

Associative Classes



Generalization and Specialization

It is sometimes the case that two or more classes:
– have common attributes
– participate in the same relationships
– have common behaviour
To avoid duplication in our models, we use generalization and specialization.

3.4 Attributes

3.5 Attributes Constrain

4 State Modelling

Life-cycle modelling
We will use state machines to model the life-cycles of

- **Real-world things**
- **Real-world relationships**

State Machines are often associated with development of real-time systems

- – Process Control
- – Telephone switches
- – Household appliances
- – washing machines, dryers, DVD players
- – Vending machines
- – Control systems
- – flight, car cruise, robotics

Because, real-time systems
– Act in response to external stimuli
– Act in accordance with current state
– Act in a deterministic manner
But, many non-realtime systems also need to do these things. eg.
– Business processes
– Transactions of various kinds

4.1 The components of State Machine

- **States**

Waiting for alarm to be set
Alarm activated

- **Events**

Set alarm

Clear alarm

- **Transitions**

Paths from one state to another

Associated with events

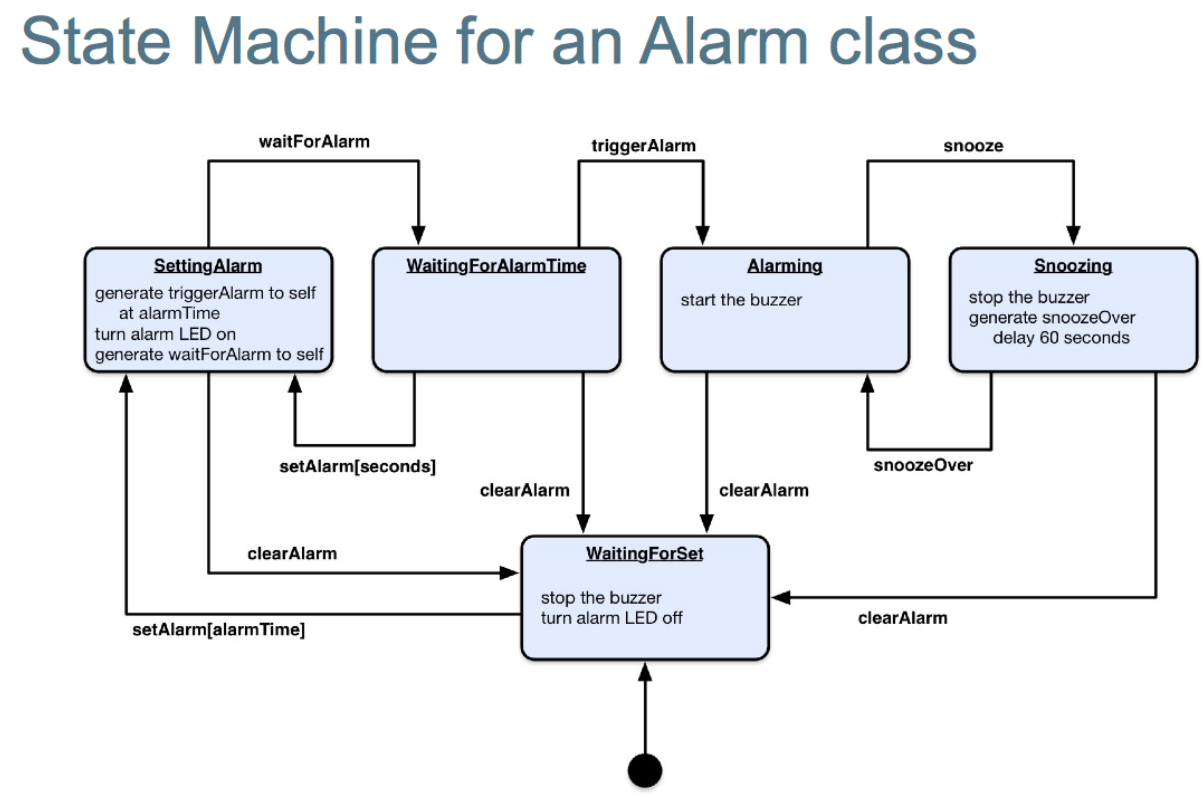
Taken when a corresponding event is received

- **Actions**

Associated with states or transitions

Executed when state changes

States:	**The rectangle**
Transitions:	**Line with arrow**
Events:	**String on the line with arrow**
Actions/ Procedures:	**String in the rectangle**



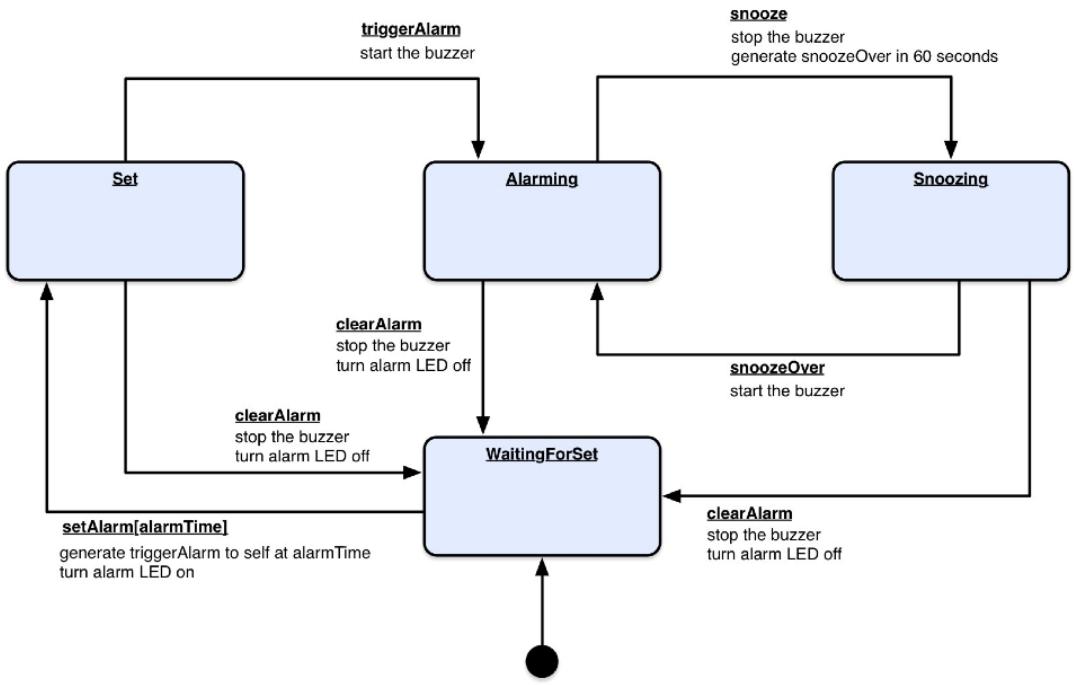
4.2 The expressions of state machine:

1. The Moore model (as in the previous slide) – Actions are associated with states (Show as above) – Actions are executed on entry to states.
1. The Mealy model

– Actions are associated with transitions – Actions are executed during transition (Show as below)

1. A combination of the abov

The Mealy model



Mealy vs Moore – which one to use?

They are equivalent, however in the alarm clock example

– **Mealy** has only 4 states, but duplicated actions

– **Moore** has 5 states, but NO duplication of actions

4.3 Two way to represent State Machine:

State Charts (Show as above)

– Provide a good diagrammatic overview of a state machine

State Transition Tables (Show as below)

– Can be used to analyse the completeness of a State Machine

Microwave Oven – State Transition Table

		Events			
		button Pressed	*door Opened*	*door Closed*	*timer Expired*
Current State	ReadyTo Cook	Cooking	Door Open	?	?
	Cooking	?	Cooking Interrupted	?	Cooking Complete
	Cooking Complete	?	Door Open	?	?
	Cooking Interrupted	?	?	Ready To Cook	?
	Door Open	?	?	Ready To Cook	?

What about the empty cells?

The empty cells need to be analysed

This analysis can identify

– Missing States

– Missing Transitions

– More questions for stakeholders

If nothing is missing, we must state if the subject event

– Is ignored (If it happens nothing, the action is ignored.)

– Can't Happen

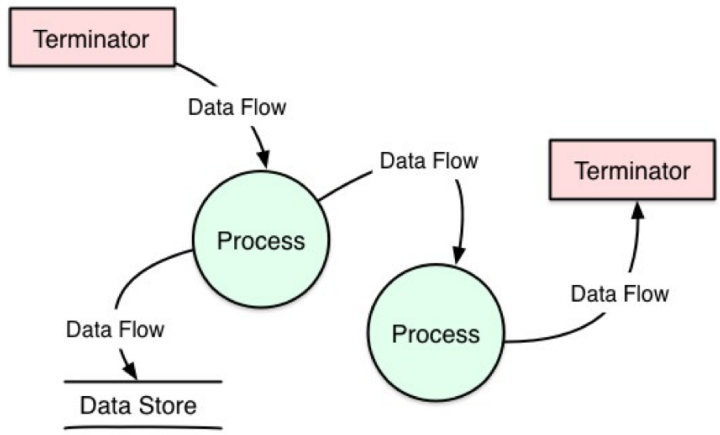
	buttonPressed	doorOpened	doorClosed	timerTimesOut
Ready To Cook	Cooking	Door Open	***Can't Happen***	Event Ignored
Cooking	Cooking Extended	Cooking Interrupted	***Can't Happen***	Cooking Complete
Cooking Complete	Cooking	Door Open	Can't Happened	***Can't Happen***
Cooking Interrupted	Event Ignored	***Can't Happen***	Ready To Cook	Event Ignored
Door Open	Event Ignored	***Can't Happen***	Ready To Cook	***Can't Happen***
Cooking Extended	Cooking Extended	Cooking Interrupted	***Can't Happen***	Cooking Complete

Figure 9.9. State Transition Table with "Can't Happen" Entries

5 Process Modelling

Data Flow Models is Not part of Executable UML or UML, But useful when modelling data processing intensive domains. It Models the data flow, transformation and storage of data in a domain, and it is Normally used to model entire systems.

Terminology and Representation



5.1 Processes

- Transform input items into output items
- Can be mechanical, human or software

- Named with a short action clause
- Summarise what is done to the input items in order to produce the output items (not how)
- Processes are activated when information or materials are available on each of their input data flows

5.2 Data Flows

- Indicate the content and direction of flow of information (or materials) between processes, stores and terminators
- Think of them as pipelines along which items of data and materials can flow
- Names reflect their content

5.3 Data Store

- Represent data flows that are frozen for a indeterminate time
- The information and materials they represent can be accessed at any time and in any order
- Names reflect their content

5.4 Terminator

- Represent things that are external to the domain or system under consideration
- They are important because they provide and/or receive system input and output

6 executable specification

6.1 Executable model

we should execute and debug models
Just like we execute code to locate errors and determine correctness
Why executable specification?

Coding	Modelling
Integrated development environment	Modelling tool
Compiler	Model translation tools
Code debugger	Model debugger

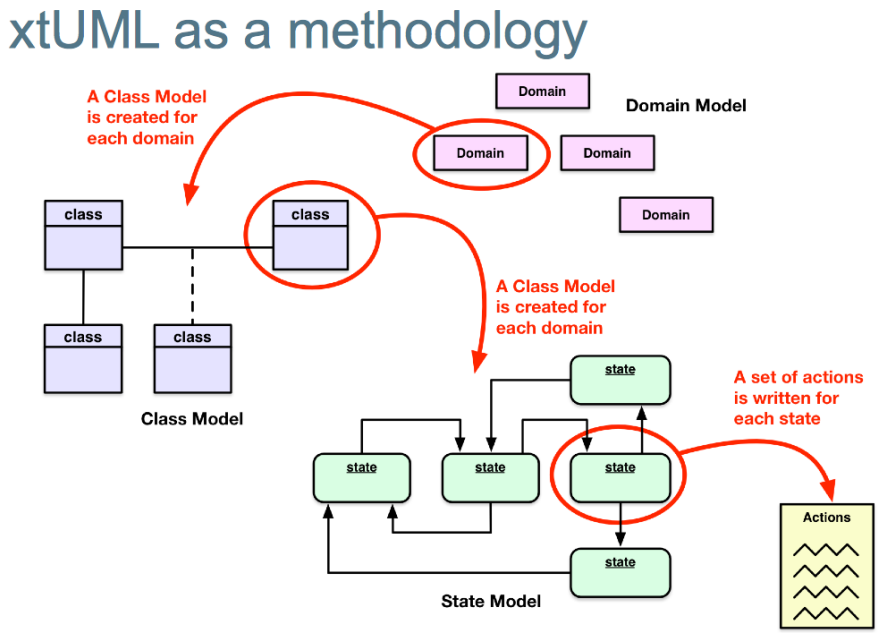
6.2 xtUML = eXecutable and Translatable UML

= UML - (Semantically weak and unnecessary element) + precisely defined action semantics

- xtUML is language
    - Simple, coherent and sufficient subset/superset of UML

(Class models, State Models, Action Language, Object Constraint Language)

- Executable
- xtUML is also a methodology
    - A process for building software systems
    - Based on the separation of concerns



Question 1:

What is the purpose of the Object Constraint Language?
Select one:
a. It is used to add detail to UML models.
b. It is used to write state actions.

UML models are often not able to express all that is known about a domain. The OCL can be used to add such detail to UML models.
The correct answer is: It is used to add detail to UML models.

Question 2:

Which of the following is used to model the behaviour of a domain?
Select one or more:
a. State models associated with active classes
b. Communication between passive classes
c. Communication between the environment and passive classes
d. Domains
e. Communication between active classes
f. State models associated with passive classes
g. Communication between the environment and active classes

Active classes are those classes in a domain that have behaviour.
Passive classes do not have behaviour - they only have attributes andassociations.

Active classes can communicate with each other and the external environment.
The way in which an active class responds to events is defined in state models.

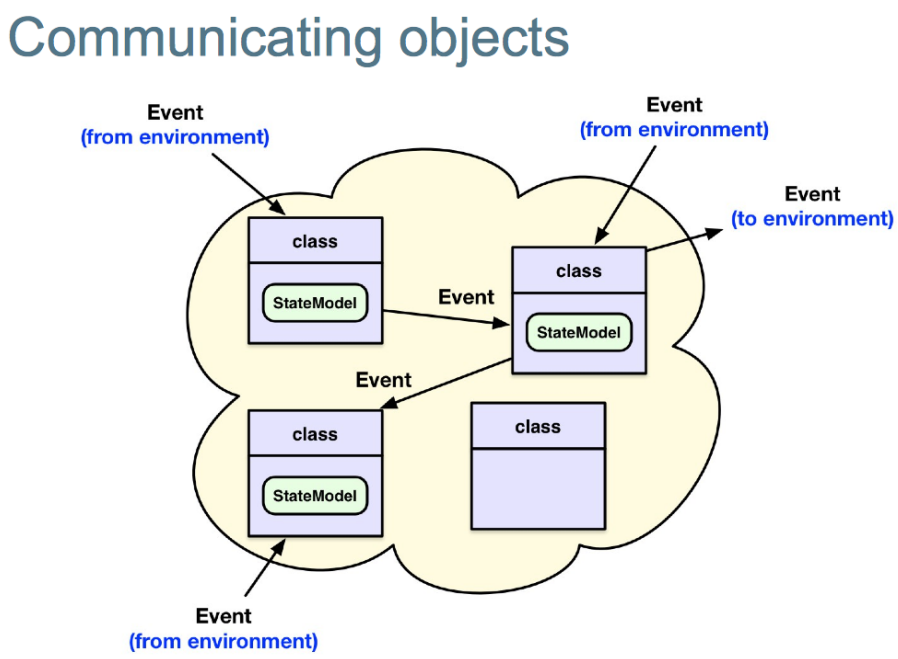
The correct answer is: Communication between active classes, Communication between the environment and active classes, State models associated with active classes

6.3 Communicating objects

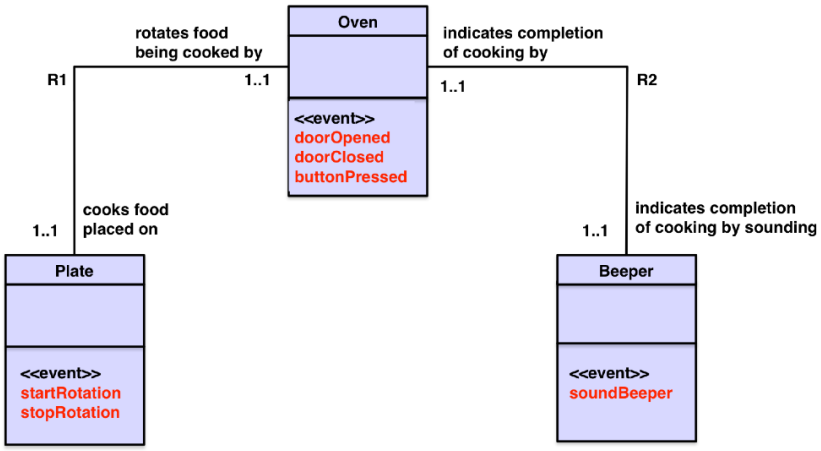
The behavior of a domain is described by
– A set of state models
– Events passed between state models
– Events passed between state models and actors in the external environment

6.4 How object Communicate?

- Objects communicate by sending signals
  - Objects send signals to other objects
  - These signals are perceived as events by the receiving object
- More generally, an event is anything that can be detected by an objects state machine
  - Signals from other objects
  - Time events
  - A condition in the external environment
- State actions can be used to send signals
  - Note: Classes can not send events. They send signal which is perceived as events by the receiving object.
- Signalling is **asynchronous, NOT synchronous.**
  - Sender continues processing after a signal is sent
  - Sender does not wait for signal to be received
- Depending on the state of the receiver object, the event may cause a state transition or it may be ignored



Communicating objects – oven example



6.5 During the advanced lectures...

- Event parameters
- Action language – Generating signals
  - Signal timing
  - Signals to/from external environment
  - Traversing links (instances of associations)
  - Creating and deleting objects and links
- Visualising domain dynamics – Collaboration diagrams
  - Sequence Diagrams
- Model Verification
  - Static verification
  - Dynamic Verification
  - Test cases

Question 3:
During the advanced lectures we will cover *static* and *dynamic* model verification. What is *static* verification? Select one: a. A process used to check that a model has the expected behaviour b. A process used to check that a model complies with the xtUML rules and that class models and state models reflect what is known about a domain. c. A process used to check that a model complies with the xtUML rules. Static analysis checks all aspects of a model without executing it. This includes checking that class models and state models reflect what is known about a domain. Static analysis also checks that the models are constructed in accordance with the xtUML rules.

The correct answer is: A process used to check that a model complies with the xtUML rules and that class models and state models reflect what is known about a domain.