Interaction Diagrams I

Lecture # 4

Department of Computer Science and Technology
University of Bedfordshire

Written by David Goodwin,
based on the book *Applying UML and Patterns (3rd ed.)*
by *C. Larman* (2005).

Modelling and Simulation, 2012

Outline

Interaction Diagrams

Communication Diagrams

- information space
- stereotypes
- Example #1
- Traceability
- Procedure
- Example #2

Interaction Diagrams

Interaction Diagrams

- Interaction diagrams illustrate how objects interact via messages.
- Used for **dynamic object modeling**.
- Two common types:
  - Communication Diagrams
    - illustrate object interactions in a graph or network format; objects can be placed anywhere on the diagram.
  - Sequence Diagrams
    - illustrate object interactions in a "fence" format; each new object is added to the right of the diagram.

Interaction Diagrams

- The act of dynamic interaction modeling is incredibly valuable.
- Spend time doing dynamic object modeling with interaction diagrams, not just static object modeling with class digrams.
  - need to think of
    - What messages to send?
    - Messages to whom?
    - Messages in what order?

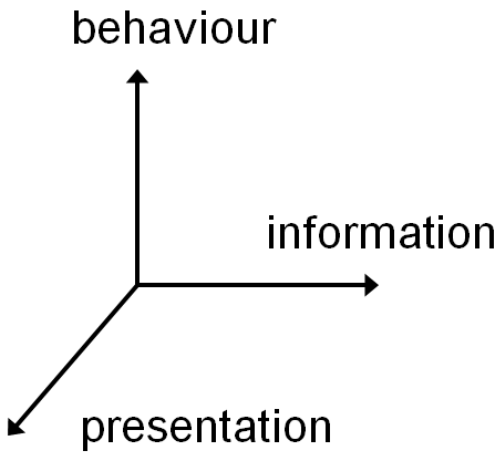
Notation - Lifeline Boxes

- Lifeline boxes represent the participants in the interaction
- Participants are often instances of a class
  - Below left; lifeline box representing an unnamed instance of the class "Sale".
  - Below right; lifeline box representing a named instance of the class "Sale".

Communication Diagrams

Communication Diagrams

- Information space
  - what information a system holds, which describes the system's internal states
  - what behaviour a system will adopt, which determines the state transition
  - what a system presents to the outside world
- an object in an information space can play one of three different roles, that is, holding information, changing states, and interfacing



Stereotypes

- An 'Analysis Class' represents an abstraction of one or more classes and/or subsystems (Use Cases) in the systems design.
  - Typically you use two modelling techniques, communication and sequence diagrams to model analysis classes.
    - An analysis class can define attributes (if needed).
    - An analysis class will include associations.
    - An analysis class will consume operations.
    - There are three stereotype classes (boundary, control & entity).

SEMANTICS

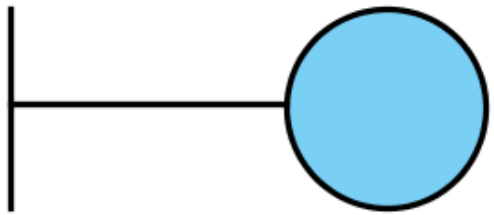
- Semantics
  - Entity object models information. It holds information and some operations that naturally related to the information.
  - Boundary/interface object models input and output information and operations that process the information.
  - Control object models functionality/operations that process information from several different entity objects.

PRAGMATICS

- Pragmatics
  - Identifying interface objects - functions directly related to actors.
  - Identifying entity objects - information used in a Use Case and functions of processing information
  - Identifying control object - functions that unite interface objects and entity objects
  - (Carry out the identifications for EVERY use case defined)

BOUNDARY CLASSES

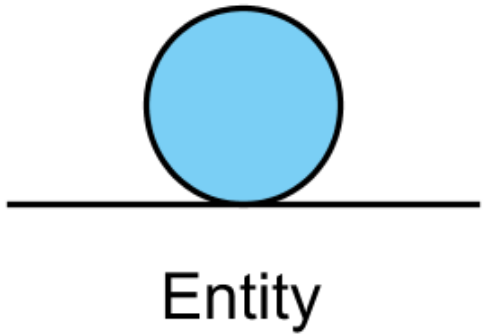
- This is a class used to model the interaction between the system and its actors.
- The interaction often involves inputs and outputs.
- They clarify and collect information regarding the systems' boundary (user interface)
- Boundary classes represent abstractions of windows, or other communication interfaces etc. (the computer system).
- Each boundary class should be related to at least one actor and vice versa.



Boundary

ENTITY CLASSES

- An entity class is used to model persistent data (information) AND business logic.
- An entity class is usually not passive.
- An entity 'object' can show changes in data(information).
- Entity classes often show a logical data structure ready for design.
- Entity classes also show what data(information) the system is dependent upon.



CONTROL CLASSES

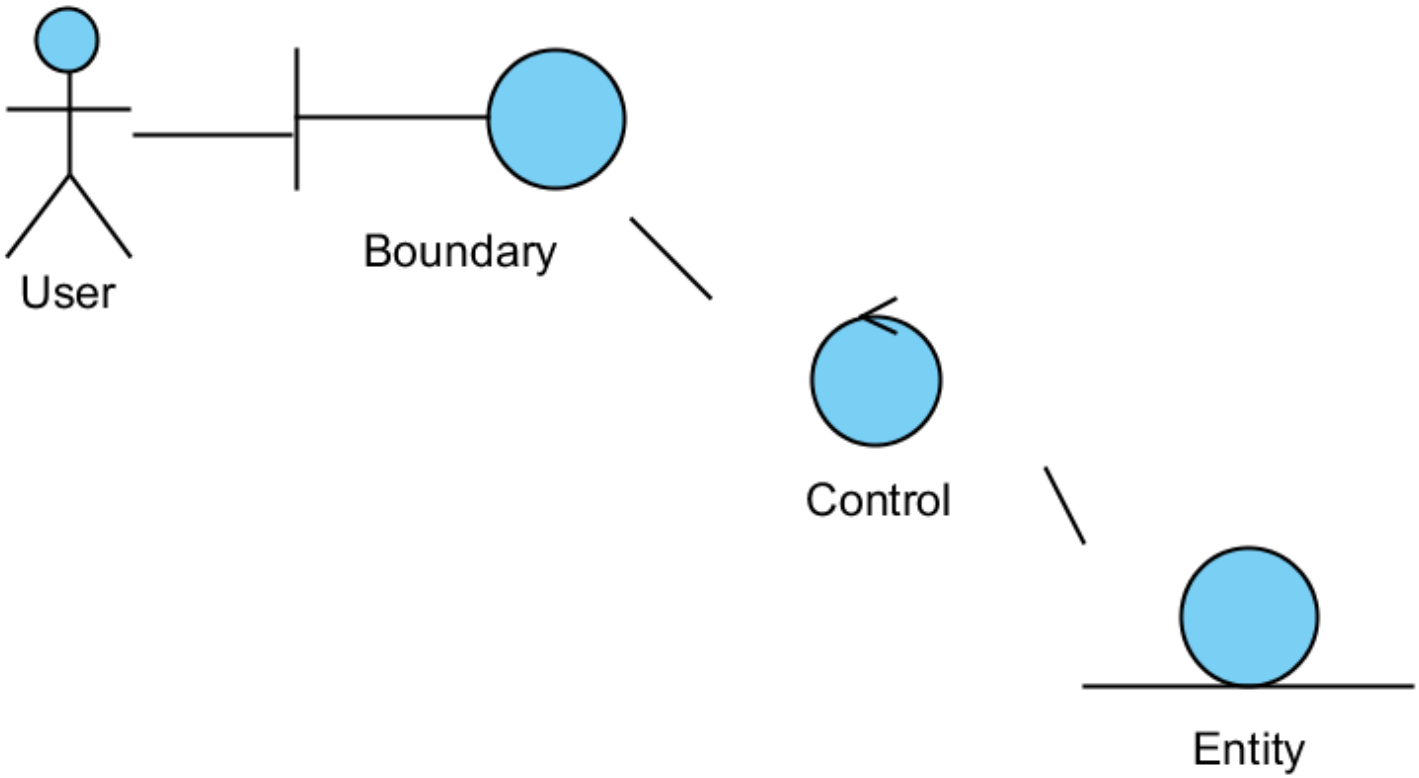
- Control classes represent coordination, sequencing, transactions and control of other objects.
- The dynamics of the system are modelled by control classes.
- Typically they coordinate the movement (parameters) between boundary and entity classes.

Control

Identifying Analysis Classes

- Examine the models of a Use Case and identify (for a communication diagram):
  - entity classes by examining the data(information) storage requirements of the Use Case.
  - one central boundary class for each actor. (primary window).
  - one control class responsible for handling the control of the Use Case (and introduce further control classes as necessary, for example UI handling).

'Architecture' of a Communication Diagram



Return Item Use Case in recycling machine - example

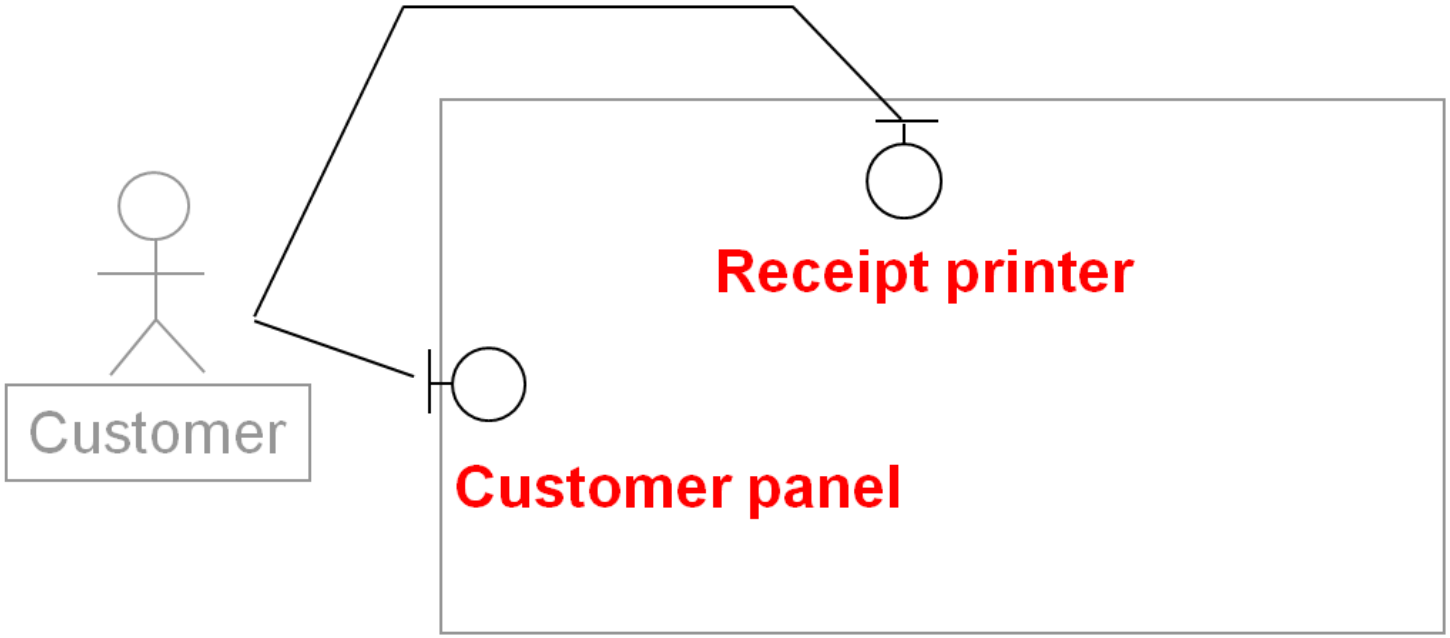
- Identification of boundary, control and entity stereotypes are often tangled together
- We normally start from the refinement of the Use Case diagram.
- Then we identify objects.
- At the "same time", we organise and identify relationship.

Identify boundary/interface objects

- Identify boundary/interface objects from linguistic description: looking for key words

"When a customer returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has been returned. If acceptable, the total number of items of this type returned by the customer increments. If not, the light for "Not Valid" is highlighted on customer panel.
When the customer presses the receipt button, the printer prints the date. The total number of items he returned and the lump sum is calculated. The following is printed out: customer number, number returned, deposit value, total of this type and lump sum"

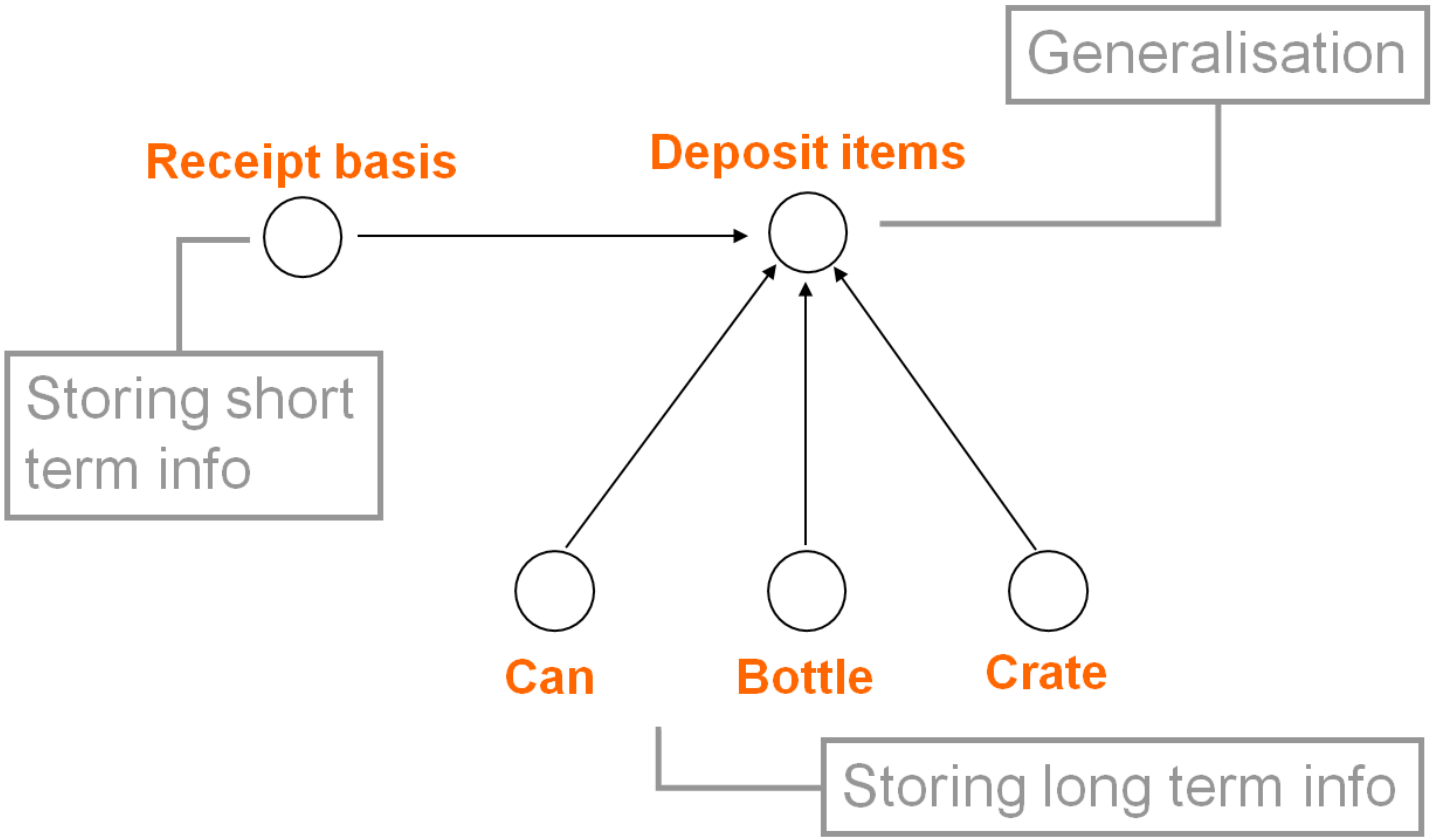
Boundary objects



Identify entity objects

- Identify entity objects from linguistic description
  - long term information (for all customer):
    - deposit values of bottle, can and crate
  - short term information (for each individual customer):
    - the total number of items of each type returned by the customer,
    - the total number of items he returned and the lump sum that should be paid to him,
    - date

Entity objects



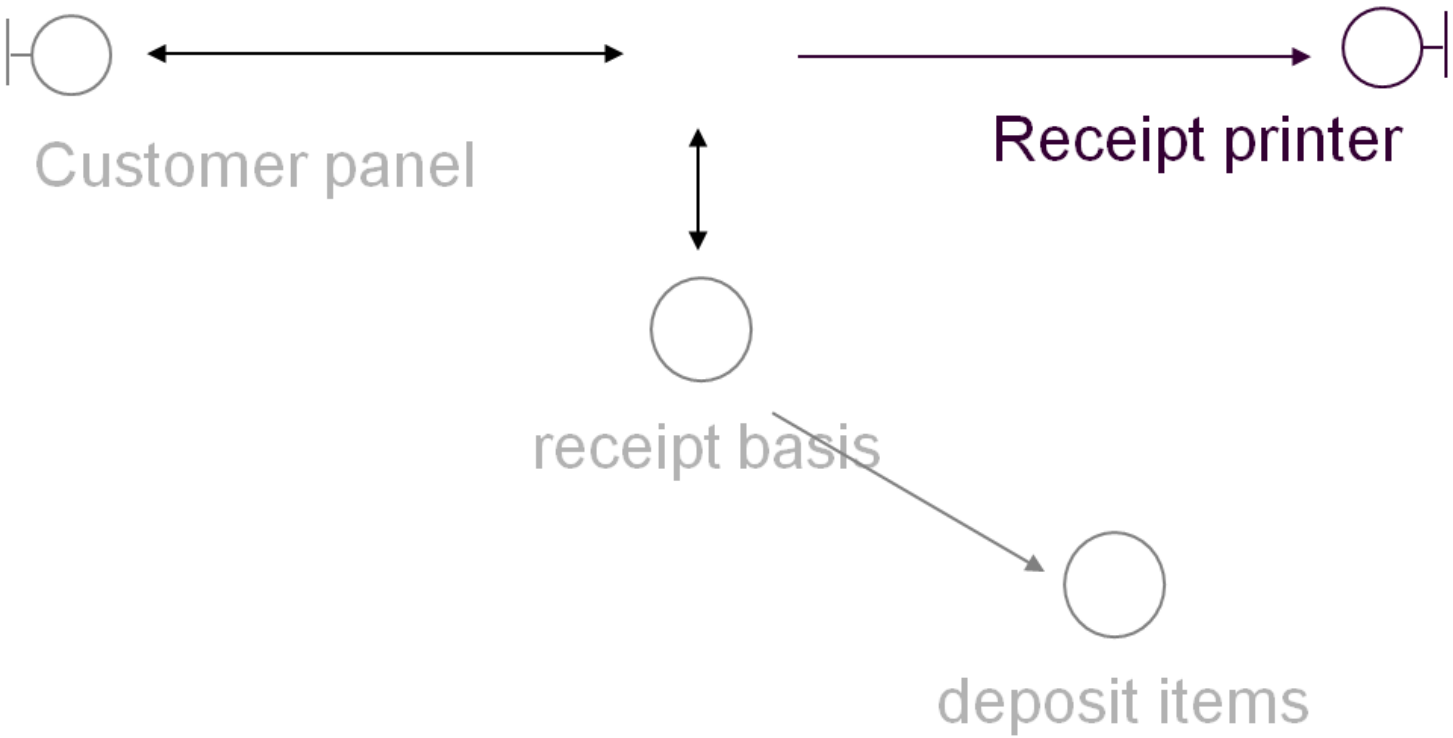
Identify control objects

- Identify control objects from linguistic description
  - There are entity and boundary objects in this use case.
    - Items coming from customer panel need to be measured and decision on whether passing the information and passing what information needs to be made according the measurement.
    - Decision on whether print out information also needs to be made.

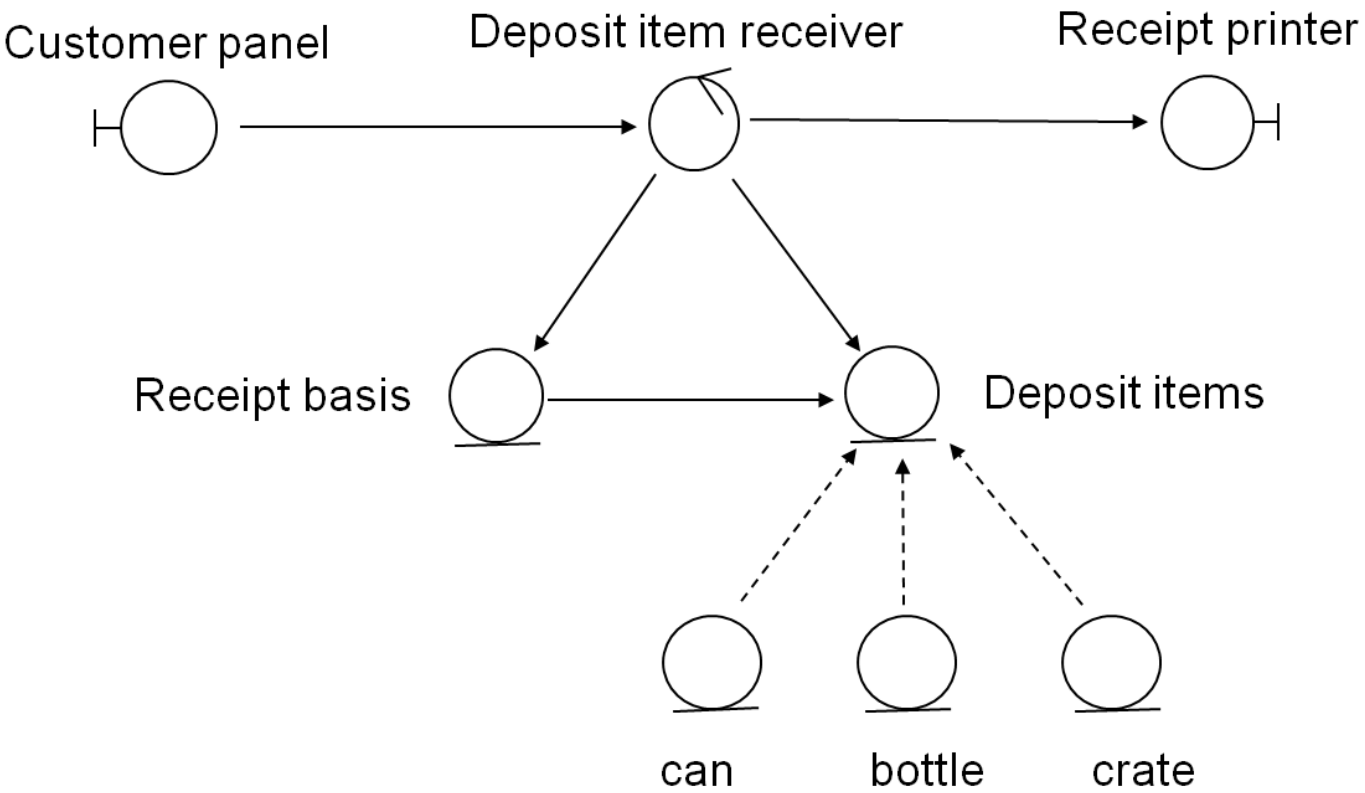
Return Item Use Case in recycling machine - example

"When a customer returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has been returned. If acceptable, the total number of items of this type returned by the customer increments. If not, the light for "Not Valid" is highlighted on customer panel.
When the customer presses the receipt button, the printer prints the date. The total number of items he returned and the lump sum is calculated. The following is printed out: customer number, number returned, deposit value, total of this type and lump sum."

Control objects



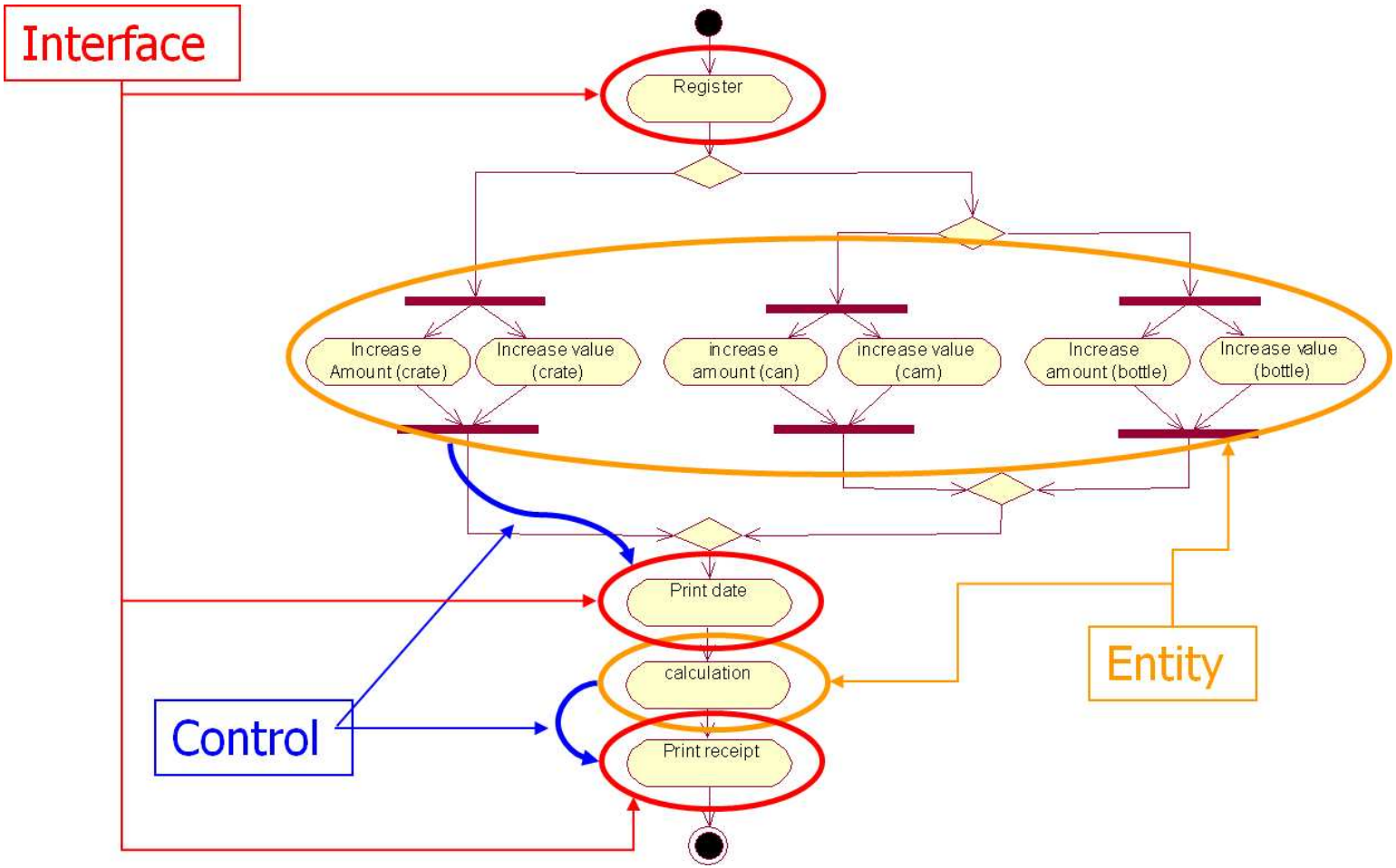
Put everything together



Return Item Use Case in recycling machine - example

- Identify the conceptual classes from activity diagram
  - interface: which activities interface actors?
    - Register is triggered by inserting deposit items
    - Print is triggered by pressing print button
  - Entity: which activities require data?
    - Increment item and increment value count values and numbers for the 3 types of items
    - Calculation requires values and numbers
  - Control:
    - Passing information is required

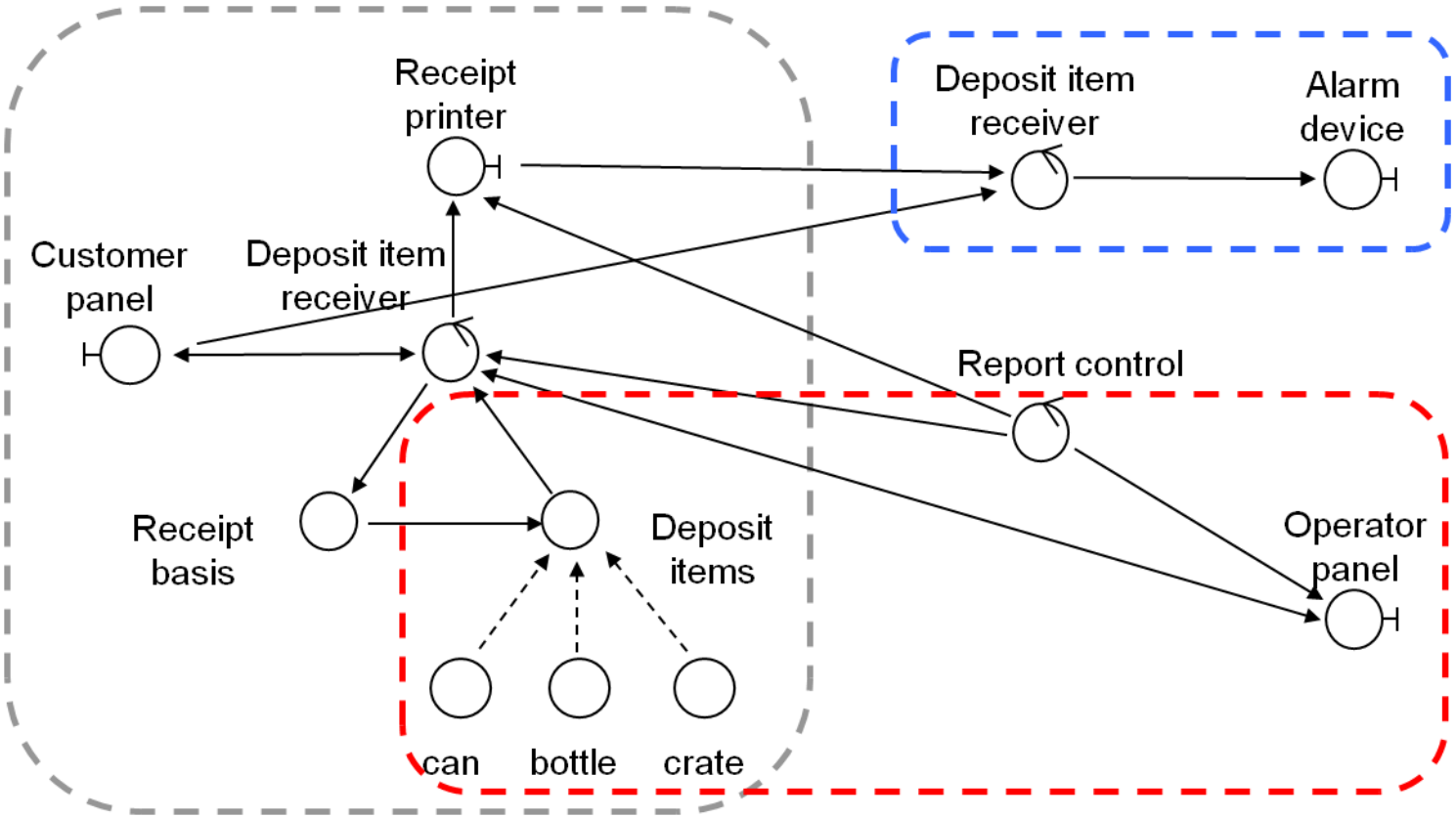
Activity Diagram



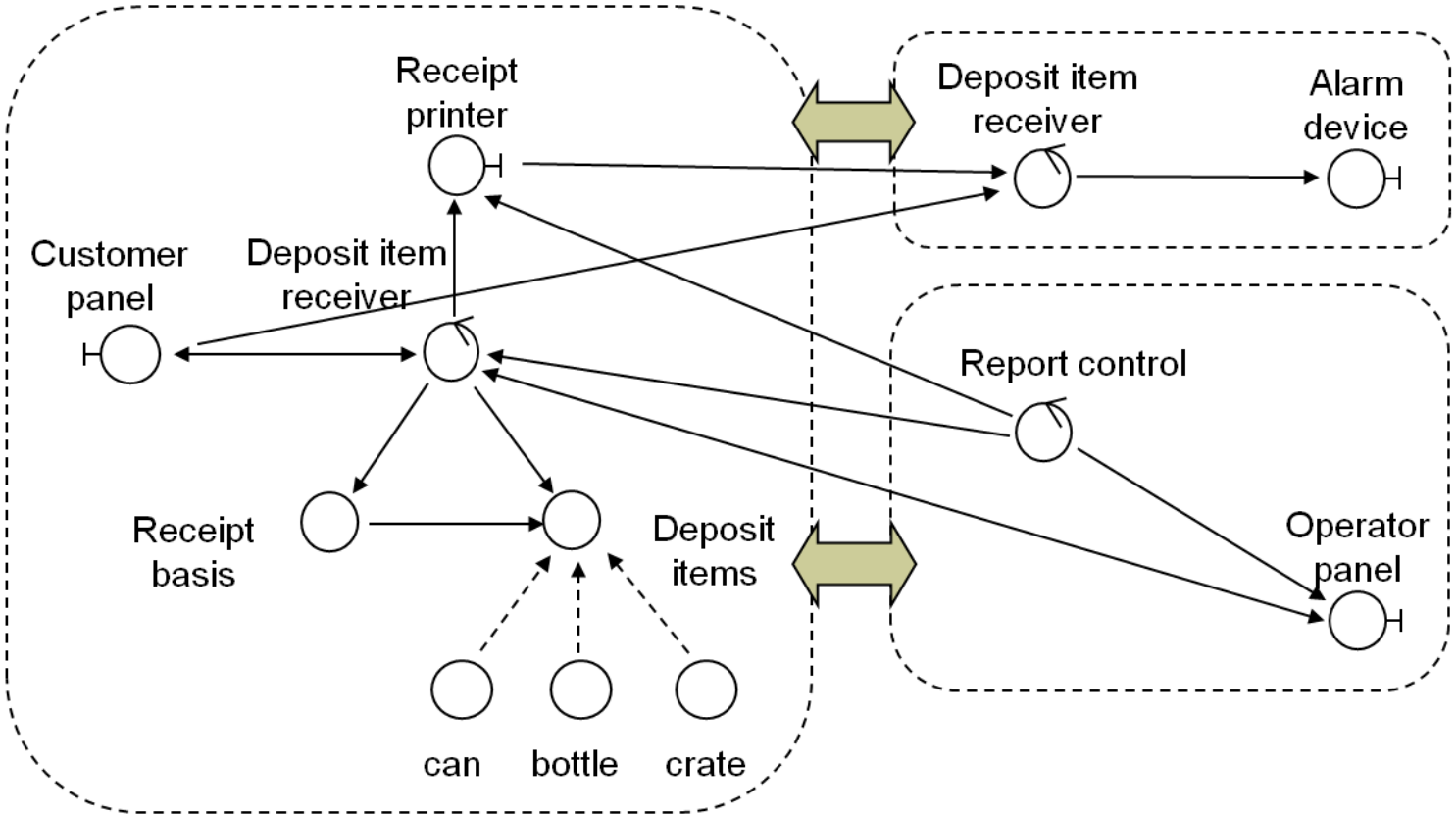
System specifications

- Logical structure of a system, i.e.
  - Components
  - Relationship between components
- An OO analysis model shows the logical structure of a system with a set of analysis classes (objects) and associations
  - analysis classes represent "components",
  - association represents relationship.
- The model also shows sub-systems and the relations between the subsystems

Analysis model of recycling machine



Three subsystems



Traceability

- It shows step-by-step how user's requirements are satisfied in the course of software development and where changes are introduced during the lifecycle of a system
- It is important for being able to trace back when we want to find places where things go wrong
- So far, we are able to trace back from OO analysis model to use case diagram as the OOAM comes from use cases.
- Traceability needs to be kept on and on in the whole course of software development

What design modeling is about

- Design aims to give a detailed description of a system to facilitate actual coding
- A design model is required to represent this description
- Actual implementation environment needs to be taken into account when building up the design model.
- Functional localisation guarantees that any functionality change will not influence large part of a system

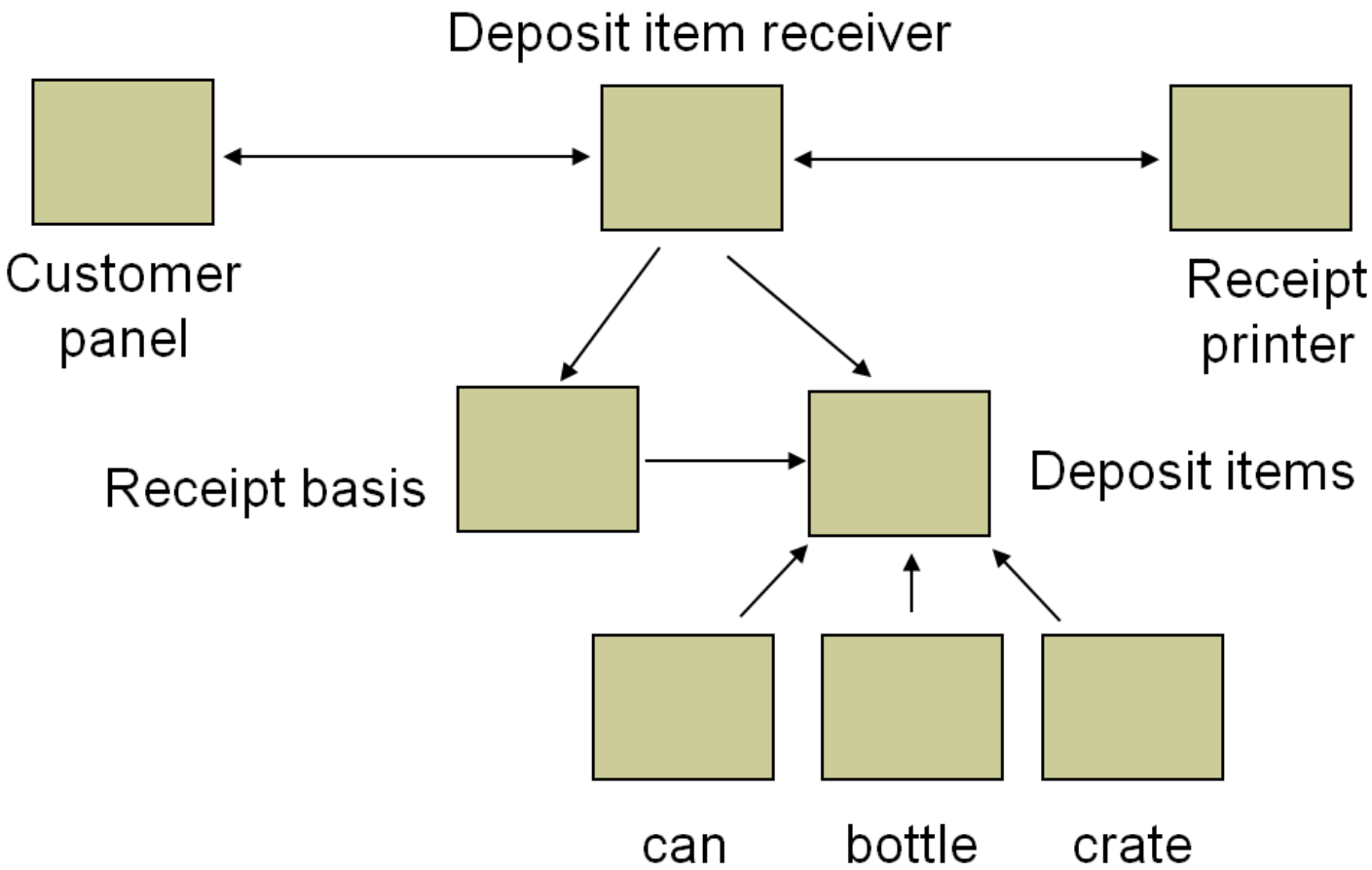
Procedure

- Converting OOAM and refining conceptual classes
- Developing sequential diagram to show information flow among the classes
- Giving details of each class
- Showing state transition of each class
- Verifying design

Conversion of OOAM

- The process of constructing a design model starts from OOAM, to guarantee the logic and traceability
- Conversion:
  - Change objects, also known as "lifelines" in Visual Paradigm, in an OOAM to "blocks"
  - Split a block into several blocks or add more blocks if necessary
- Traceability - bridging OOAM and OODM enables one to see how OODM "inherits" system's logic from OOAM
- Functional localisation - functionality is restricted in each block

Return Item Use Case in recycling machine - example



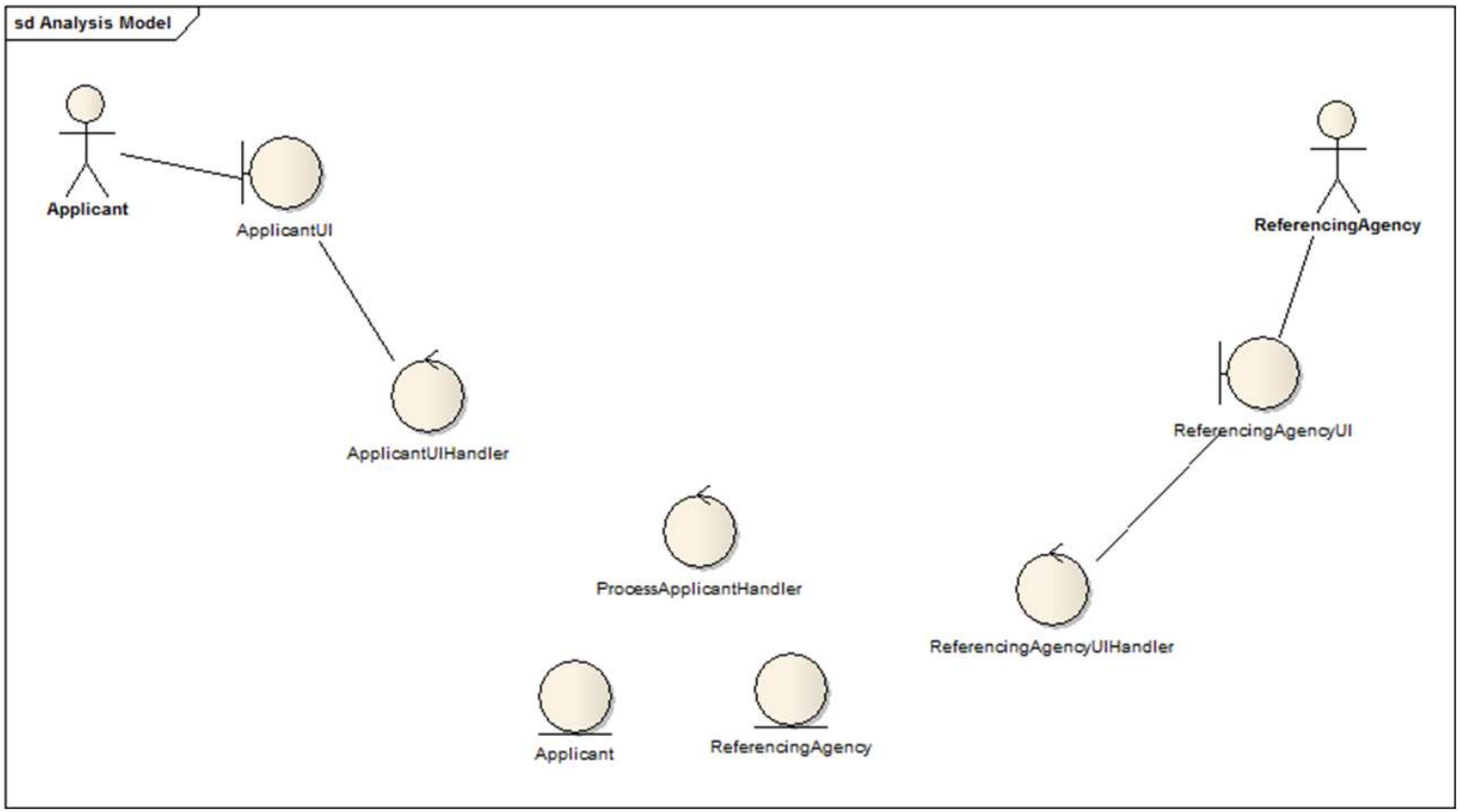
Take into account of implementation environment

- Implementation environment
  - Target environment - where system will execute
  - Programming language
  - existing products - e.g. Database
- Taking them into consideration in design
  - Encapsulation - new block will be created
  - Component-based design

Step 1

- For one of your use cases identify:
  - entity classes by examining the data (information) storage requirements of the Use Case.
  - one central boundary class for each actor. (primary window).
  - one control class responsible for handling the control of the Use Case (and introduce further control classes as necessary, for example UI handling).
- By drawing:

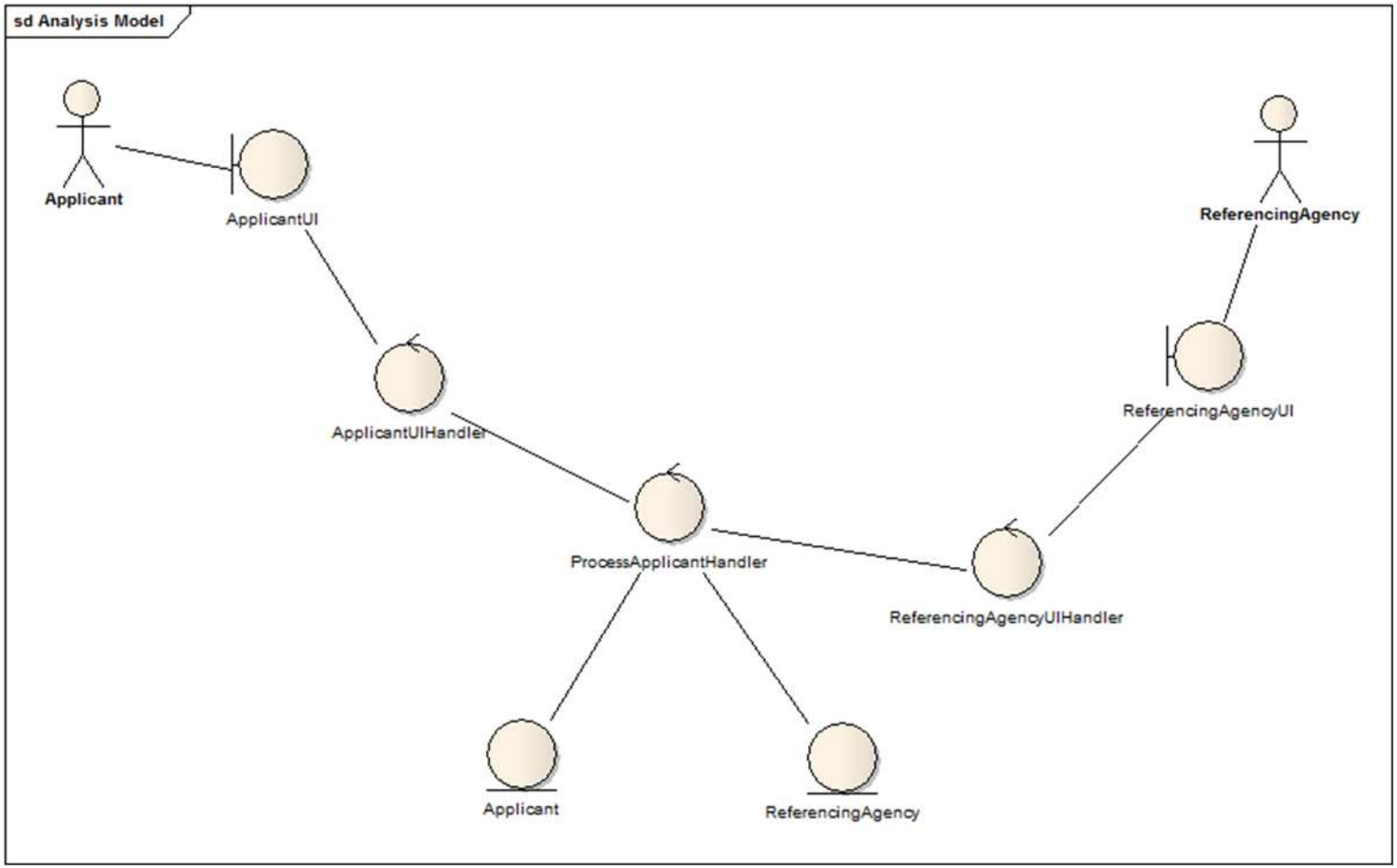
Step 1


sd Analysis Model
Applicant
ApplicantUI
ApplicantUIHandler
ReferencingAgency
ReferencingAgencyUI
ReferencingAgencyUIHandler
ProcessApplicantHandler
Applicant
ReferencingAgency

Step 2

- Add links to the diagram.
  - Links are ‘candidate’ associations. Follow the pattern of:
    - Links from actor(s) to boundary classes (objects);
    - Links from boundary classes (objects) to UI handlers;
    - Links from UI handlers to use case controller class (object);
    - Links from use case controller class (object) to entity classes (objects).

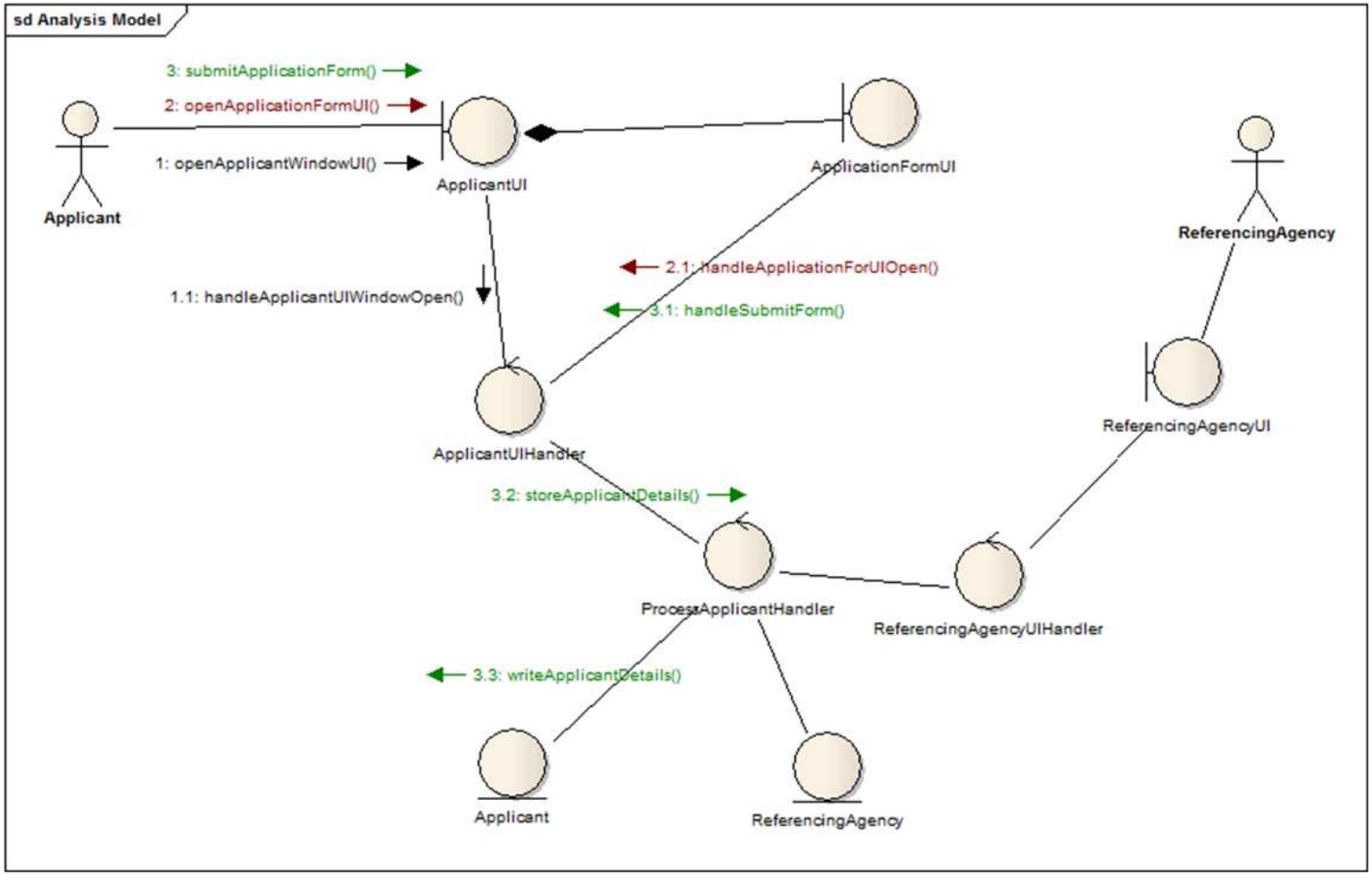
Step 2


sd Analysis Model
Applicant
ApplicantUI
ReferencingAgency
ApplicantUIHandler
ReferencingAgencyUI
ProcessApplicantHandler
ReferencingAgencyUIHandler
Applicant
ReferencingAgency

Step 3

- Add messages to the diagram.
  - Messages become candidate operations.
  - The class receiving the message takes ownership of the operation.
- Review your activity diagrams to compile a list of interactions that an actor will make with the use case.
- Write a list of interactions and number them.
- Add the list to the diagram.
  - THE DIFFICULT BIT!
- Repeat for each use case!
- example:
  1. openApplicantWindowUI
  2. openApplicantFormUI
  3. submitApplicantForm

Step 3


sd Analysis Model
3: submitApplicationForm()
2: openApplicationFormUI()
1: openApplicantWindowUI()
Applicant
ApplicantUI
ApplicationFormUI
ReferencingAgency
2.1: handleApplicationForUIOpen()
3.1: handleSubmitForm()
1.1: handleApplicantUIWindowOpen()
ApplicantUIHandler
ReferencingAgencyUI
3.2: storeApplicantDetails()
ProcessApplicantHandler
ReferencingAgencyUIHandler
3.3: writeApplicantDetails()
Applicant
ReferencingAgency