



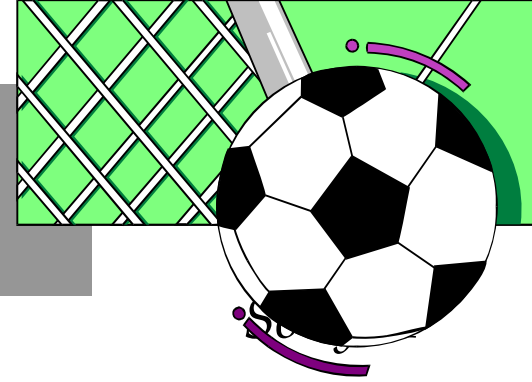
Bournemouth  
University

# Reasoning about dependencies amongst use case events

And the benefits of enaction

- Elicitation. Process models, Use Cases and interfaces.
- Writing: Using writing rules, guidelines or templates.
- Assessing Quality.
- Comprehension: Questions and interrogation
- Validation and evolution
  - *Dependencies and enactment. TOOL SUPPORT.*
- Moving towards design.
  - Teasing out (hidden) issues.
  - Use case driven processes. Construction & validation

## Two sporting use cases



1. The match reached full-time
2. The referee blew his/her whistle
3. The ball crossed the goal-line
4. The goal was not given

### **Alternatives**

4. The goal was given

1. The match reached full-time
2. The referee blew his/her whistle
3. The ball crossed the goal-line
4. The goal was given

### **Alternatives**

4. The goal was not given

Validation & Context. Someone who ‘knows the the game’.

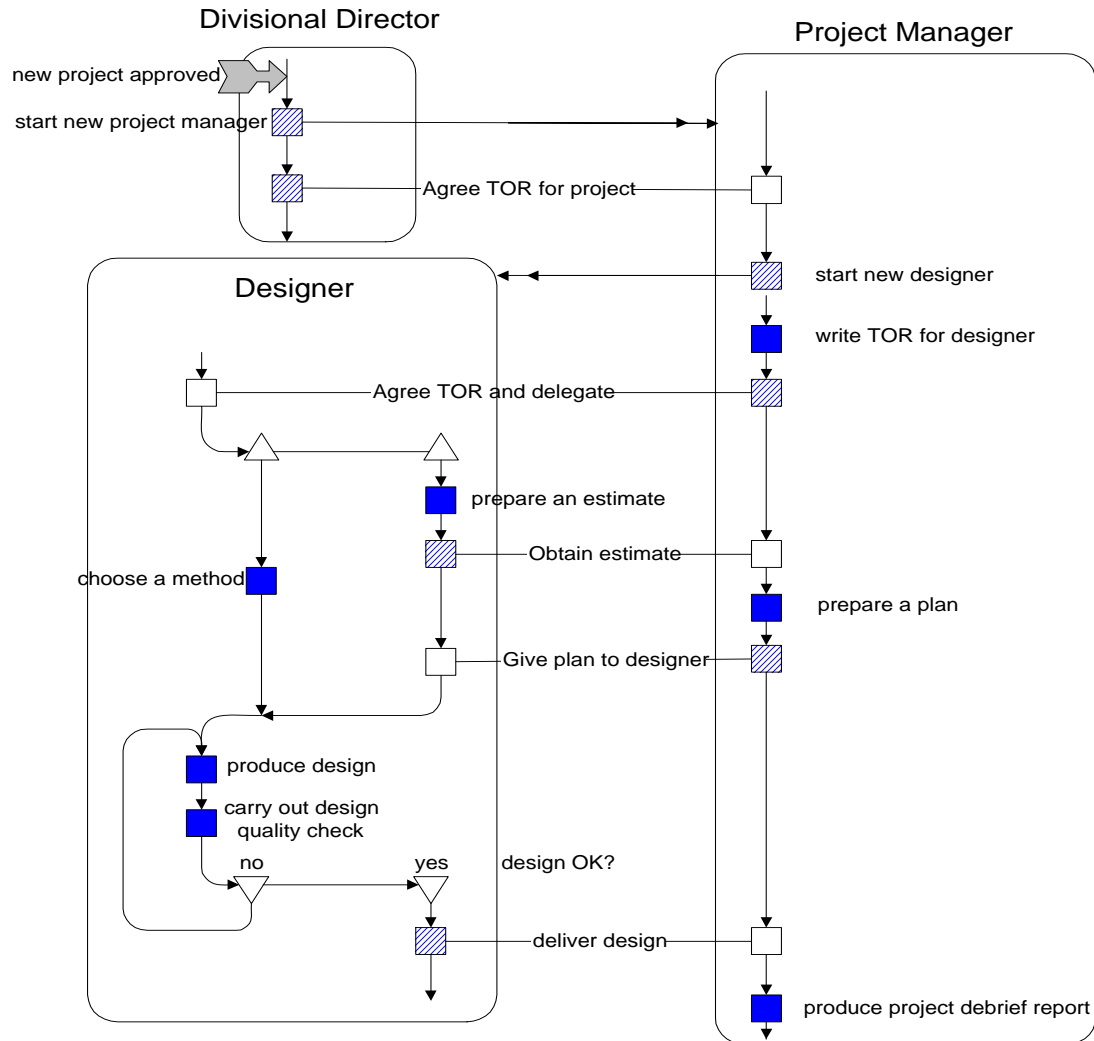
- Process modelling
- Role based models & enactable models
  - Involving *stakeholders*.
  - User-facing models. (*Audience*)
  - Industrial users: Like them but *too much effort*
- Use Cases (stuck with them)
  - Support for use case case guidelines
  - Importance of **Dependencies**
- Mapping
  - Problems moving from business models to specification – loss of ‘richness’

Business model  
(strategic)

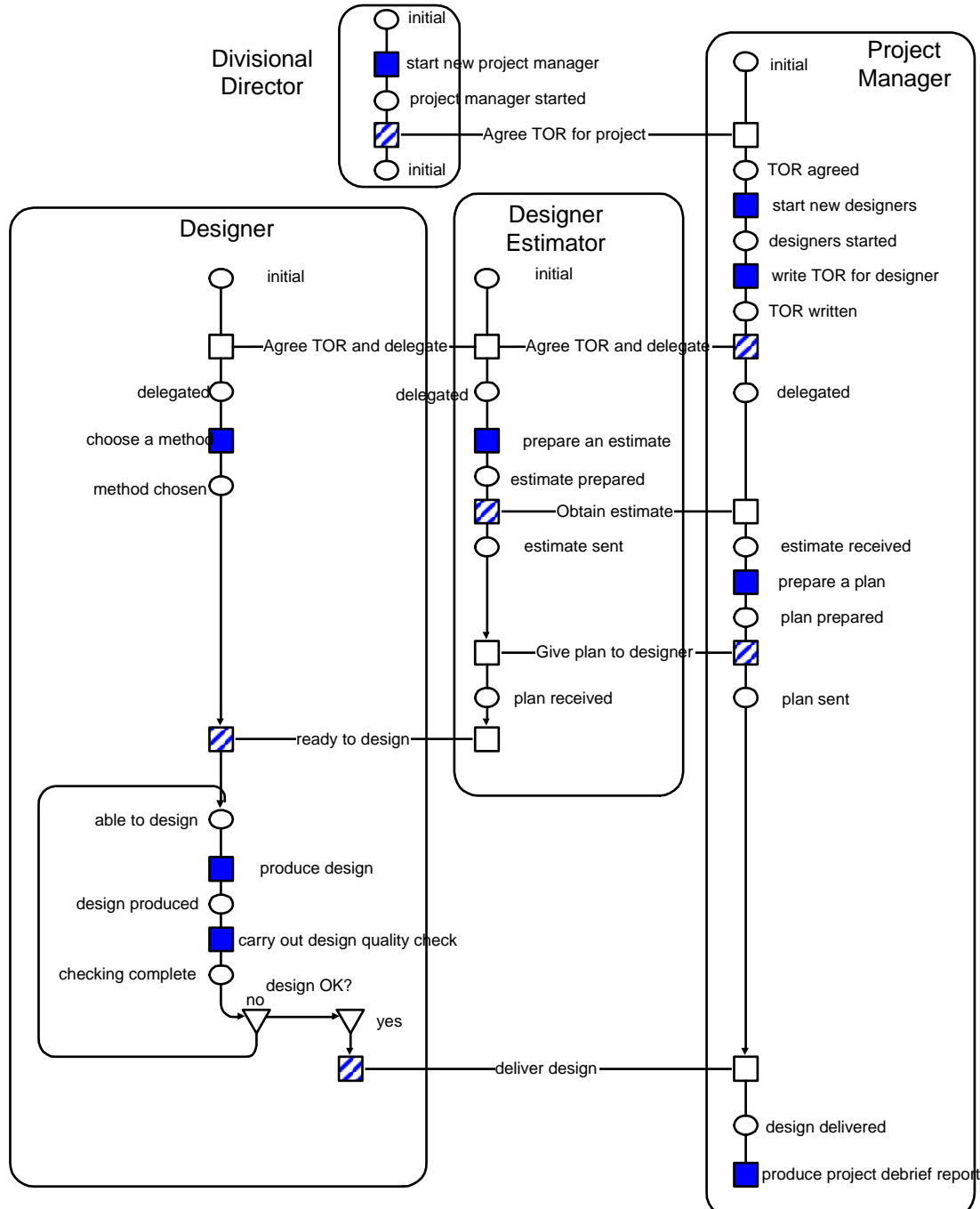
Process model  
(operational, e.g., RAD)

Use case  
(specification)

# RAD (standard)



RAD  
with  
states

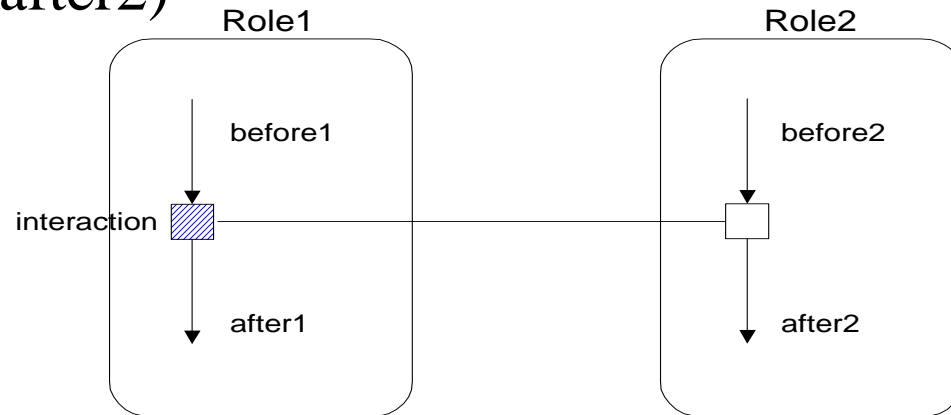


Interaction Role1.Interaction

Me(before1 → after1)

Role2(before2 → after2)

End



Interaction Designer.deliver\_design

me(accepted\_design → design\_sent)

Project\_Manager(plan\_sent → design\_received)

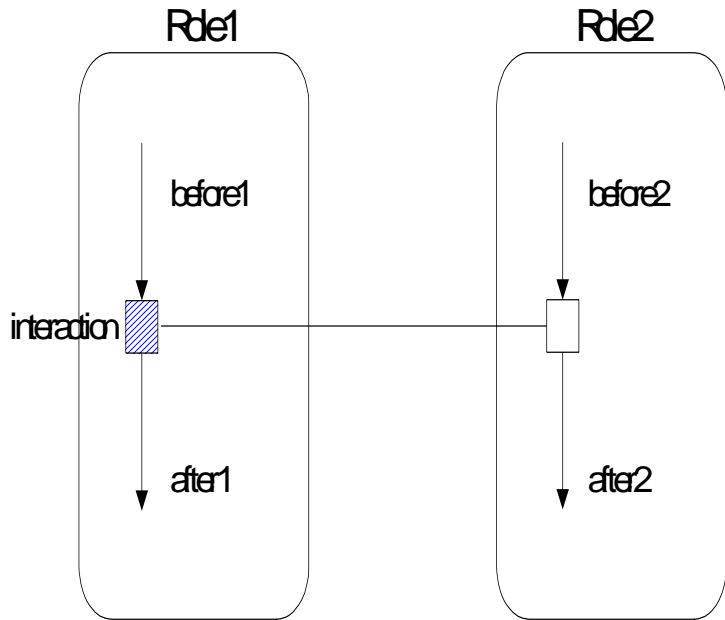
End

- Process (business) model as prerequisite to requirements or specification.
- ‘Disappointed’ by power of use case having used process models (such as RADs).
  - Enactable process models, versus static use case description.
  - Fewer options for control
  - *Information loss in moving towards specification.*
- Formal coding / annotating of use case descriptions too much effort (*especially for industrial collaborators*).



- Add pre-post to event (typically each line)
  - Interactions involve synchronisation of multiple actors.
  - Supports intra and inter-use case dependencies
  - Option to enact (order of enaction) being controlled by the pre / post *states* of events.
  - Forces consideration of dependencies amongst events.
- Allow greater stakeholder involvement.
  - Minimal (extra) effort for modeller.
- Allow traceability through from process model to use case (and beyond...)
  - Hence, don't lose the benefits.

# Three Notations



```
Interaction Role1.Interaction
Me(before1 → after1)
Role2(before2 → after2)
End
```

```
Interaction Keith.gives_pen
  Me (has_pen -> no_pen)
  Karl (no_pen -> has_pen)
End
```

**Actor**  
Keith

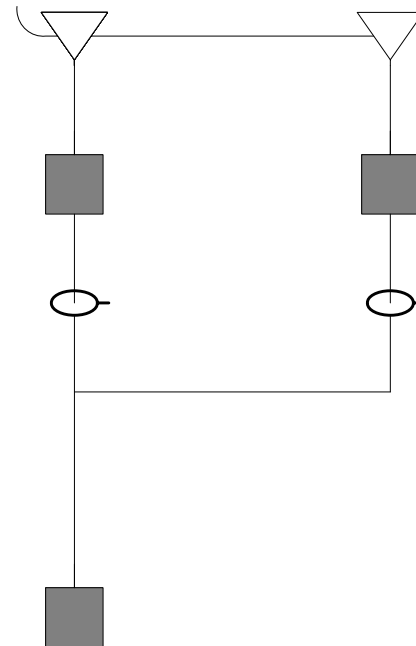
**Event**  
gives pen

<b>pre</b>	<b>post</b>	<b>Actor 2</b>	<b>pre</b>	<b>post</b>
has pen	no pen	Karl	no pen	has pen

Consider two 'independent' events, *get apples*, and *get oranges*, of some actor (or role) each which result in the post states, *has apples* and *has oranges*.

Third event, *make juice*, can occur when either apples or oranges have been obtained.

Traditionally, a guard on such an event might be a precondition such as *has fruit*.

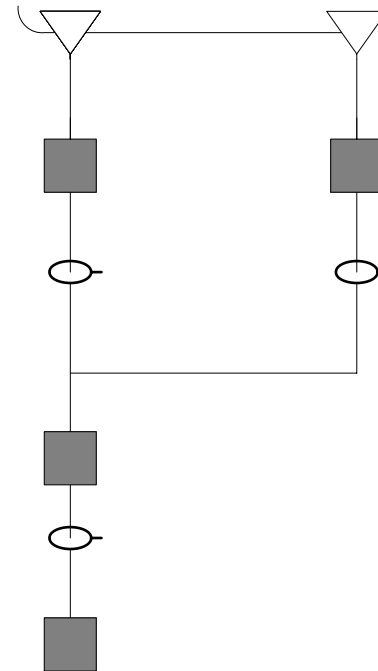


# States Only Model

Within Educator, the pre-state for make juice has to be an exact match and this requires an extra step.

The extra step brings together two threads (independent behaviours) into a single state, *has fruit*. That is, one can still arrive at the state *has fruit*, as a result of either thread. Importantly, at any given time there is still only one state for the role (or actor), and hence a further simplification, for both understanding and implementation, is preserved.

However, since state change requires an action, this means that there is a need for a further (often artificial) action in order for the actor to be in some more general state (e.g., *has fruit*).

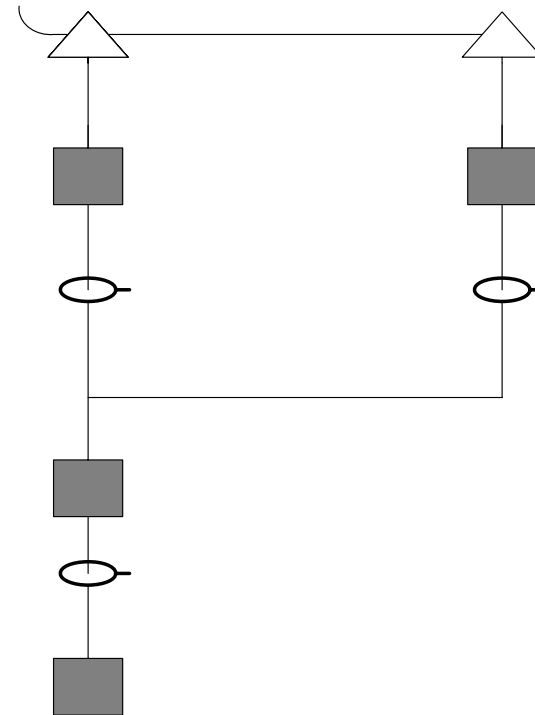


Suppose our event is now *Make smoothie*, which requires that when we have fruit. We actually have both apples and oranges.

For a use case we would be required to choose that the gaining of apples and oranges occurs in some arbitrary sequence. That is:

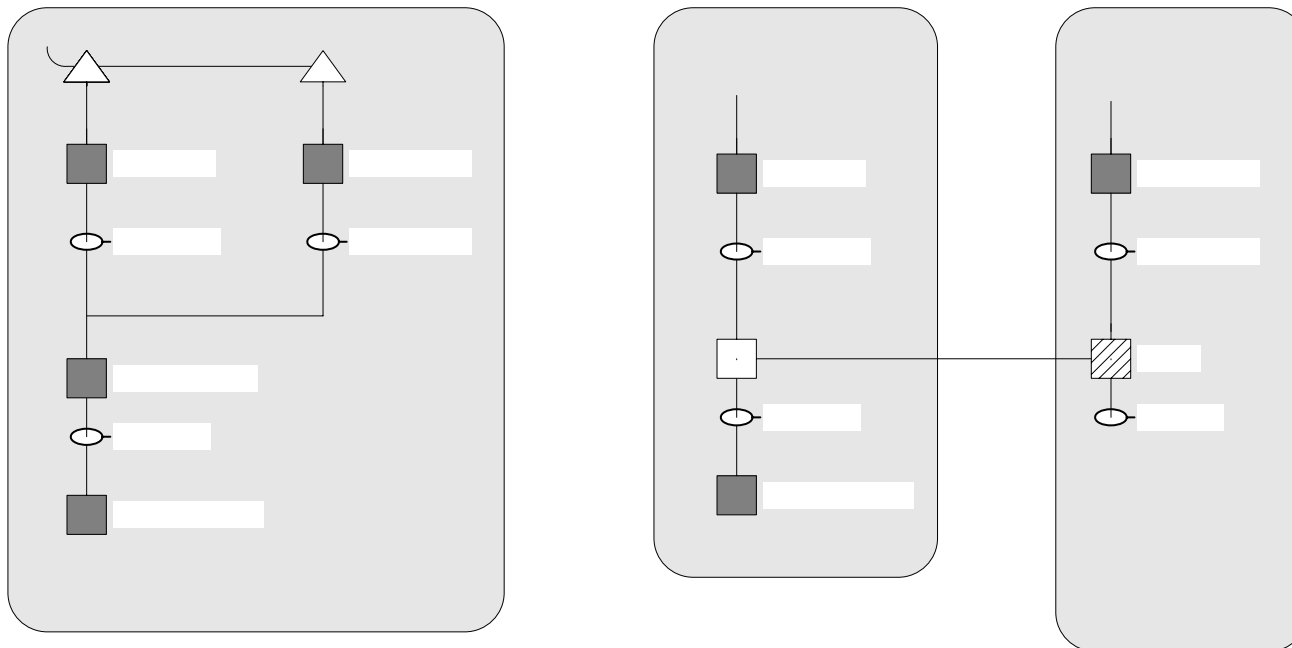
- 1 Fruit Finder get apples
- 2 Fruit Finder get oranges

However, in reality one might gather these fruits independently and in any, often unknown order.



# Parallel: Standard RAD view

We employ the mechanism of splitting the role into different roles, each of which carries one of the state variables (the having apples or oranges states).  
Below is a RAD representation of role of *Fruit Receipt* (left) and the separate roles *Apple Receipt* and *Orange Receipt* (right):



- Add pre-post to event (typically each line)
  - Interactions involve synchronisation of multiple actors.
  - Supports intra and inter-use case dependencies
  - Option to enact (order of enaction) being controlled by the pre / post *states* of events.
  - Forces consideration of dependencies amongst events.
- Allow greater stakeholder involvement.
  - Minimal (extra) effort for modeller.
- Allow traceability through from process model to use case (and beyond...)
  - Hence, don't lose the benefits.

# Strengths and Weaknesses

- Needs to spawn a role where there are multiple state variables.
- Additional roles highlight the fact that this other (independent processing) could be carried out by another resource, or may even be another role.
- Needs additional actions (or interactions) to join threads or to combine states.
- Classic precondition hides states or implies behaviour. Making states explicit forces consideration of the states of the process.
- The precondition also requires some understanding on the part of the reader (semantic load), which may not be obvious for unfamiliar models.
- A significant consideration is that the model is intended to be accessible by business users, or typical use case writers, who may not be familiar with state models (and indeed, will not carry that baggage). Hence, only one additional concept is required.



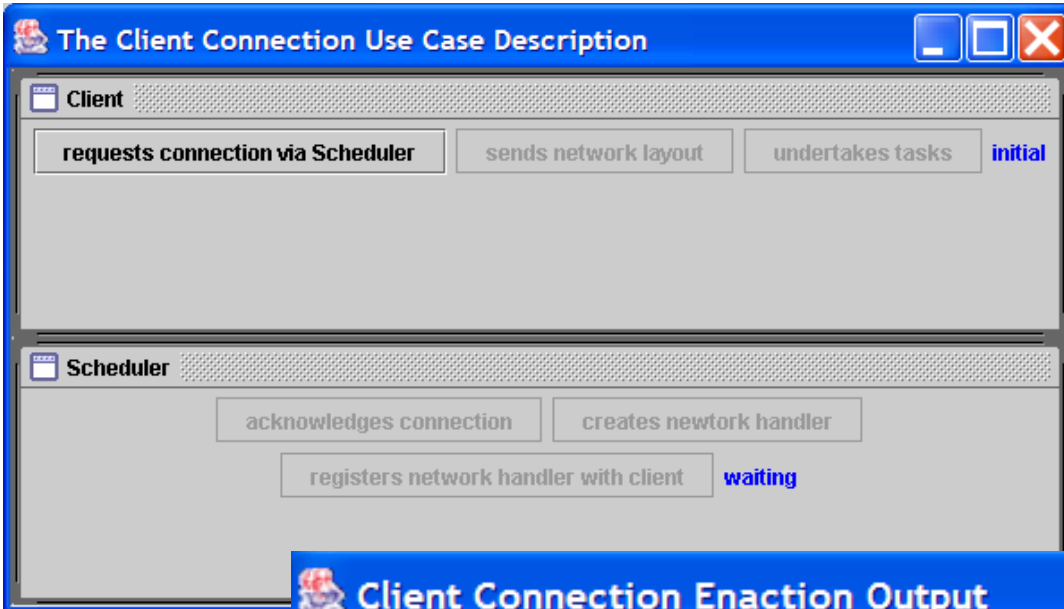
# Simple (single UC) Enaction

Editor: Use Case Enaction

File Use Case Actor Comments (next) Tools CP/Ports Help

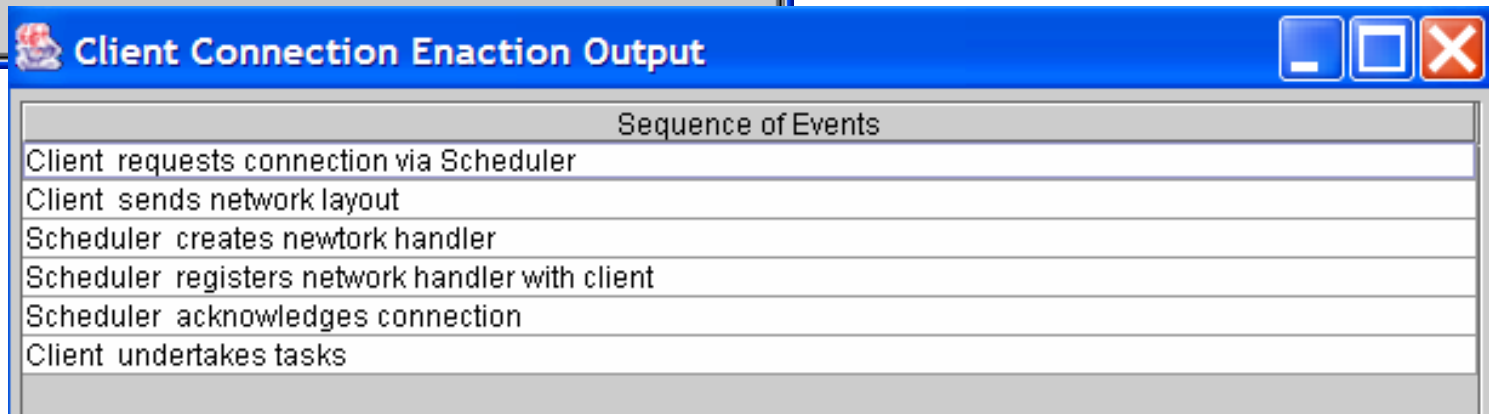
Description:

Primary Actor	Event	Precondition	Postcondition	Secondary Actor	Precondition	Postcondition
1 Lecturer	enables for courses to teach	initial	courseAgreed	Register	waiting	courseAgreed
2 Register	prepares course list	courseAgreed	initial	Student	waiting	initial
3 Student	chooses courses to study	initial	coursesChosen			



Events re-ordered. New order is in effect: 1, 3, 4, 5, 2, 6

Of course, states not written order really control invocation of events.



1. Client requests connection via Scheduler
2. Scheduler acknowledges connection
3. Client sends network layout
4. Scheduler creates network handler
5. Scheduler registers network handler
6. Client starts executing its tasks

**Educator :Use Case Enaction**

File Use Case Actor Conditions Enact Tools CP Words Help

Description: Client Connection

ID	Primary Actor	Event	Precondition	Postcondition	Secondary Actor	Precondition	Postcondition
1	Client	requests connection via Scheduler	initial	connectionRequested	Scheduler	waiting	connectionRequested
2	Scheduler	acknowledges connection	handlerRegistered	connectionAck	Client	handlerRegistered	connected
3	Client	sends network layout	connectionRequested	layoutSent	Scheduler	connectionRequested	layoutReceived
4	Scheduler	creates network handler	layoutReceived	handlerCreated			
5	Scheduler	registers network handler with client	handlerCreated	handlerRegistered	Client	layoutSent	handlerRegistered
6	Client	undertakes tasks	connected	readyToWork			

Add Description    Add Alternative path    Insert Event    Print  
 Change Precondition    Change Postcondition    Add Loop    Quit

- Consider a course registration process described with the following use case events:
  - 1. Lecturer volunteers for courses to teach.
  - 2. Registrar prepares course list.
  - 3. Student chooses course to study.

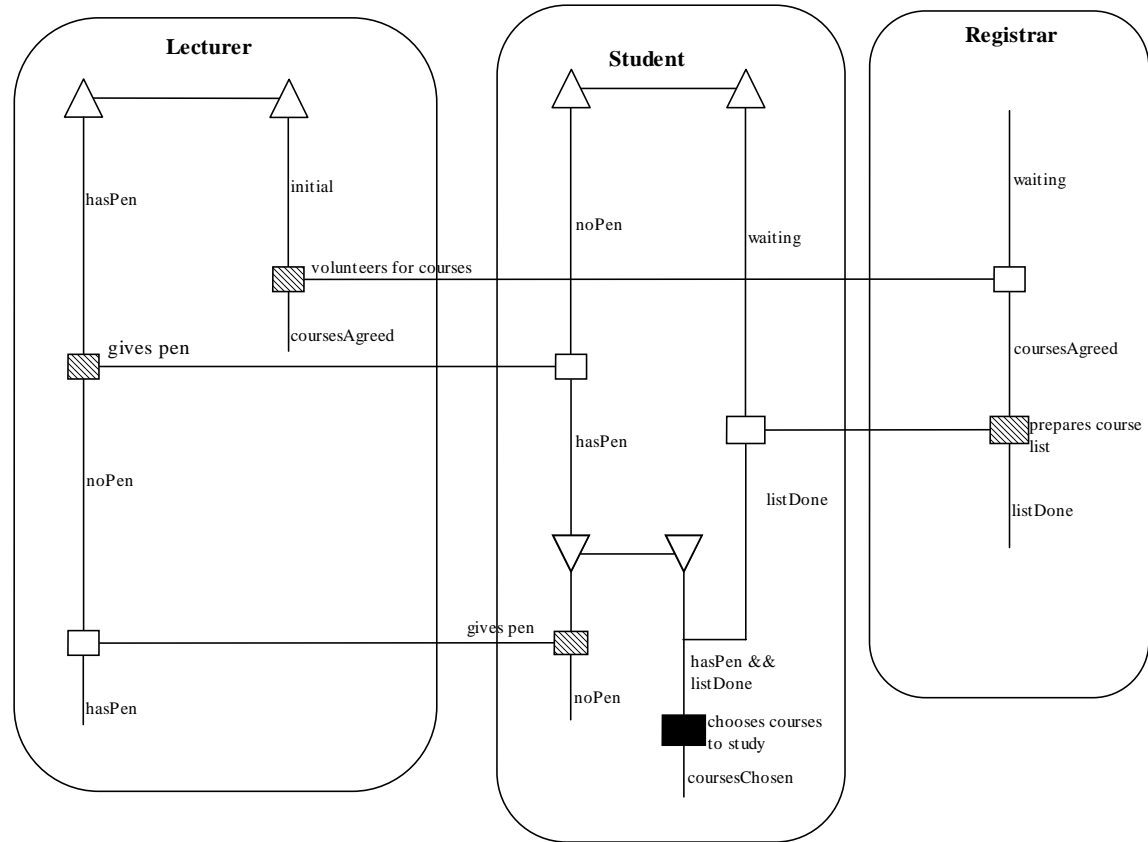
# Related Use Cases

<b>Primary Actor</b>	<b>Event</b>	<b>Pre state</b>	<b>Post state</b>	<b>Secondary Actor</b>	<b>Pre state</b>	<b>Post state</b>
Lecturer	gives pen	hasPen	noPen	Student	noPen	hasPen
Student	gives pen	hasPen	noPen	Lecturer	noPen	hasPen

<b>Primary Actor</b>	<b>Event</b>	<b>Pre state</b>	<b>Post state</b>	<b>Secondary Actor</b>	<b>Pre state</b>	<b>Post state</b>
Lecturer	volunteers for courses to each	initial	coursesAgreed	Registrar	waiting	coursesAgreed
Registrar	prepares course list	coursesAgreed	listDone	Student	waiting	listDone
Student	chooses course to study	listDone	coursesChosen			

Groupings are:

- a) **Standard roles, as for a normal RAD**
- b) By use case (ignoring roles or actors)
- c) By separating each actor that is involved in multiple use cases into separate unique roles, where each role represents that actor for a particular use case, and is named accordingly.

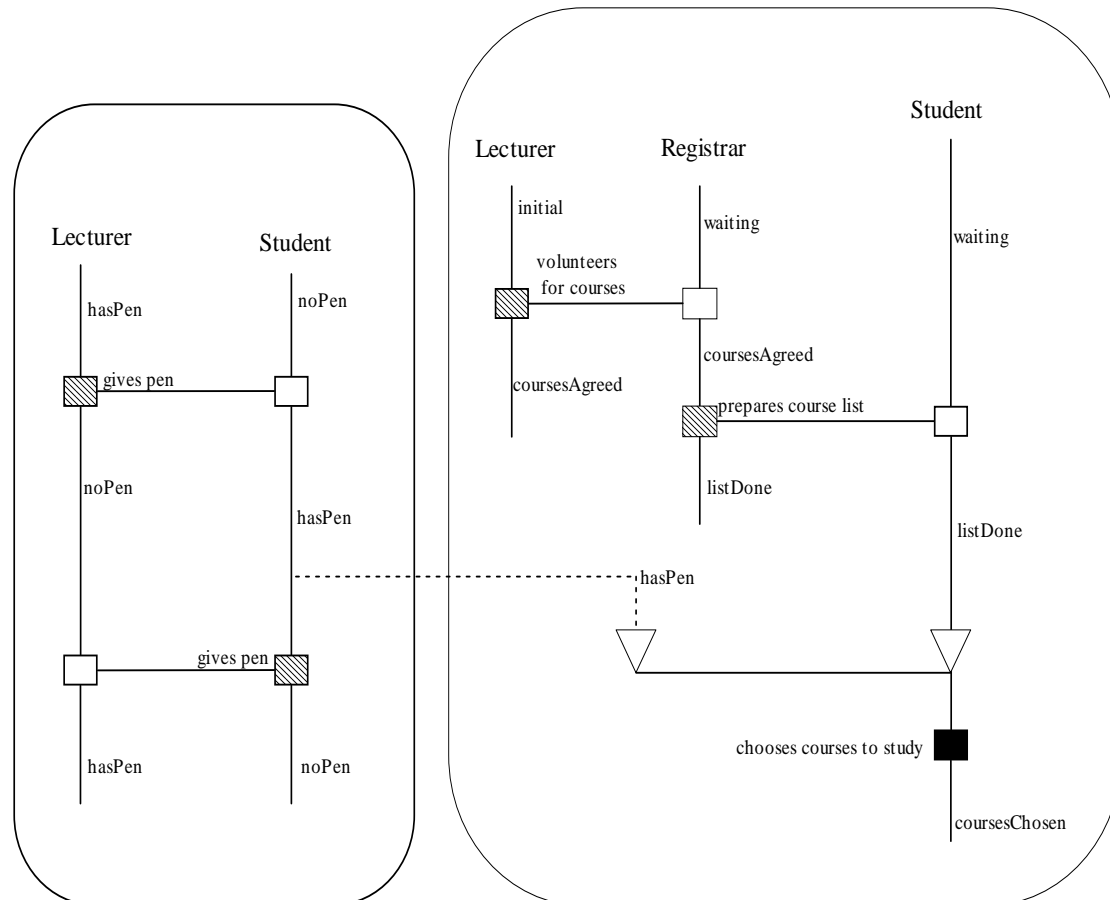


Groupings are:

a) Standard roles, as for a normal RAD

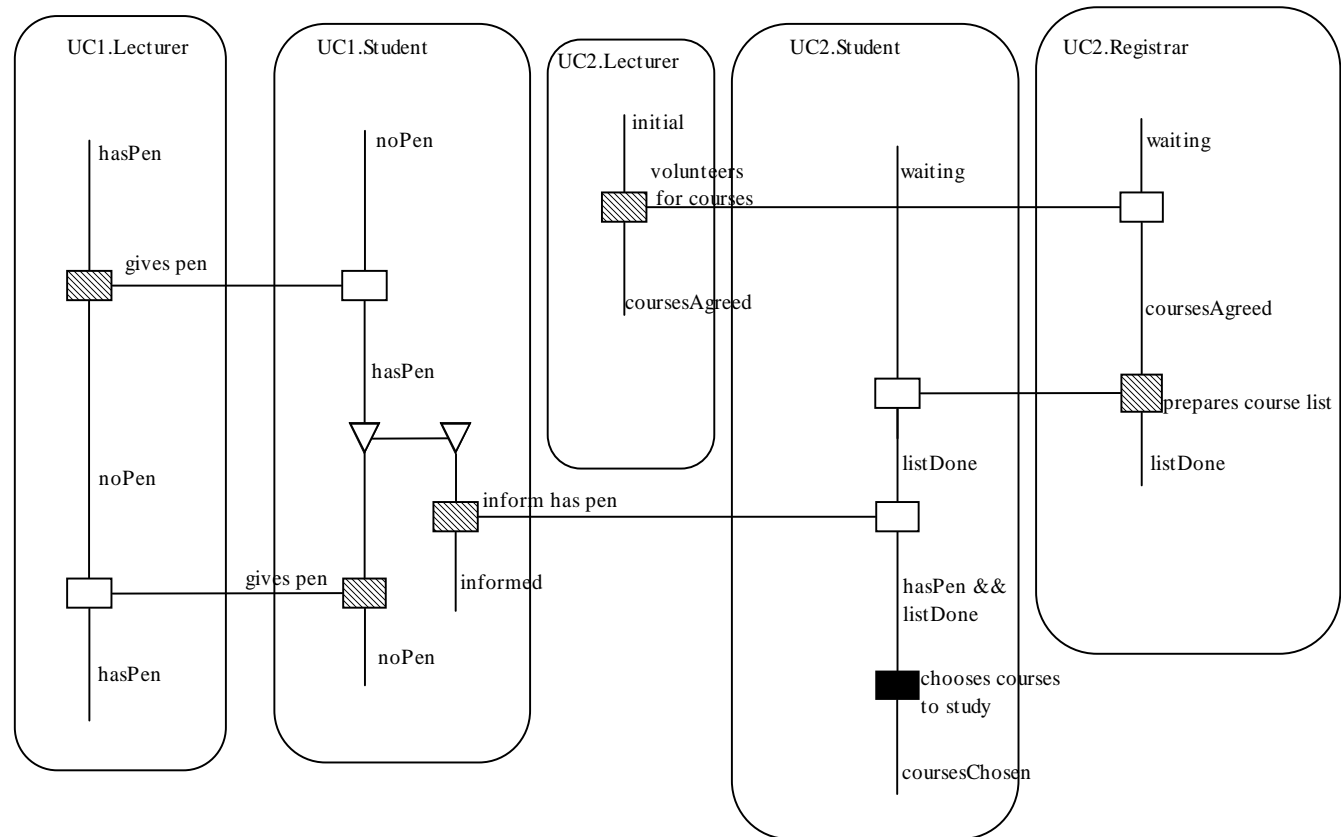
**b) By use case (ignoring roles or actors)**

c) By separating each actor that is involved in multiple use cases into separate unique roles, where each role represents that actor for a particular use case, and is named accordingly.



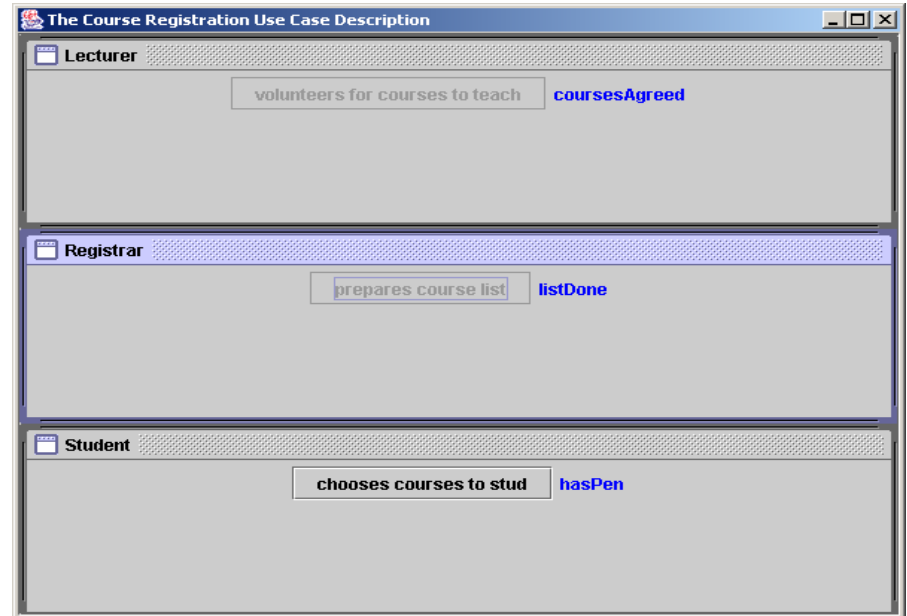
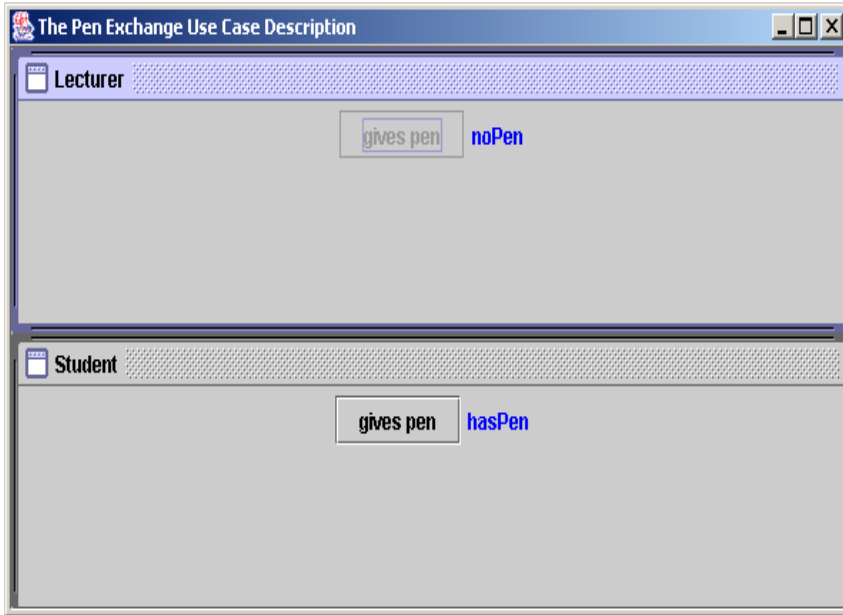
Groupings are:

- a) Standard roles, as for a normal RAD
- b) By use case (ignoring roles or actors)
- c) By separating each actor that is involved in multiple use cases into separate **unique roles**, where **each role represents that actor for a particular use case**, and is named accordingly.





# Parallel threads





Bournemouth  
University

# Findings & Conclusions



SoSyM

- Relatively easy to represent RAD with equivalent UCD
  - Maintains mapping and to aid alignment of process and use cases.
  - Though, in reality often orthogonal perspectives.
- Enaction aids discussion with stakeholders.
- Consideration of dependencies AND enaction both lead to greater *shared understanding*.
- Some process issues cannot be depicted easily.
  - Still can't represent timing or NFRs
  - Use notes indicating aspects that cannot be coded as states, actors or events.
- Even this simple augmentation to use cases can seem tricky to non-technical users

- Does the increased capability offered by dependencies enhance or overcomplicate descriptions?
- Will the inclusion of use case writing guidelines restrict the flexibility offered by enactment?
- Does the template approach to structuring use cases fit more naturally with tool support?
- Will requirements volatility make dependency mapping unmanageable?
- Do users really require models that consider dependencies across use cases, or does the restriction to consideration within a use case provide a partitioning of understanding?