Using enactable models to enhance use case descriptions

'Banging on about enaction again'

Keith Phalp

Presentation to the ESERG Workshop, July 2003

Bournemouth University, UK

Supporting Use Cases
Our (group) context

- 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 enaction. TOOL SUPPORT.*
- Moving towards design.
  - Teasing out (hidden) issues.
  - Use case driven processes. Construction & validation

Research Rationale / agenda

- Use Case Descriptions do not have good tool support.
- Validation of descriptions has always been less easy than UML suggest.
- Enaction provides an excellent opportunity to validate descriptions.
- Enaction also enables consideration of later design issues.

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

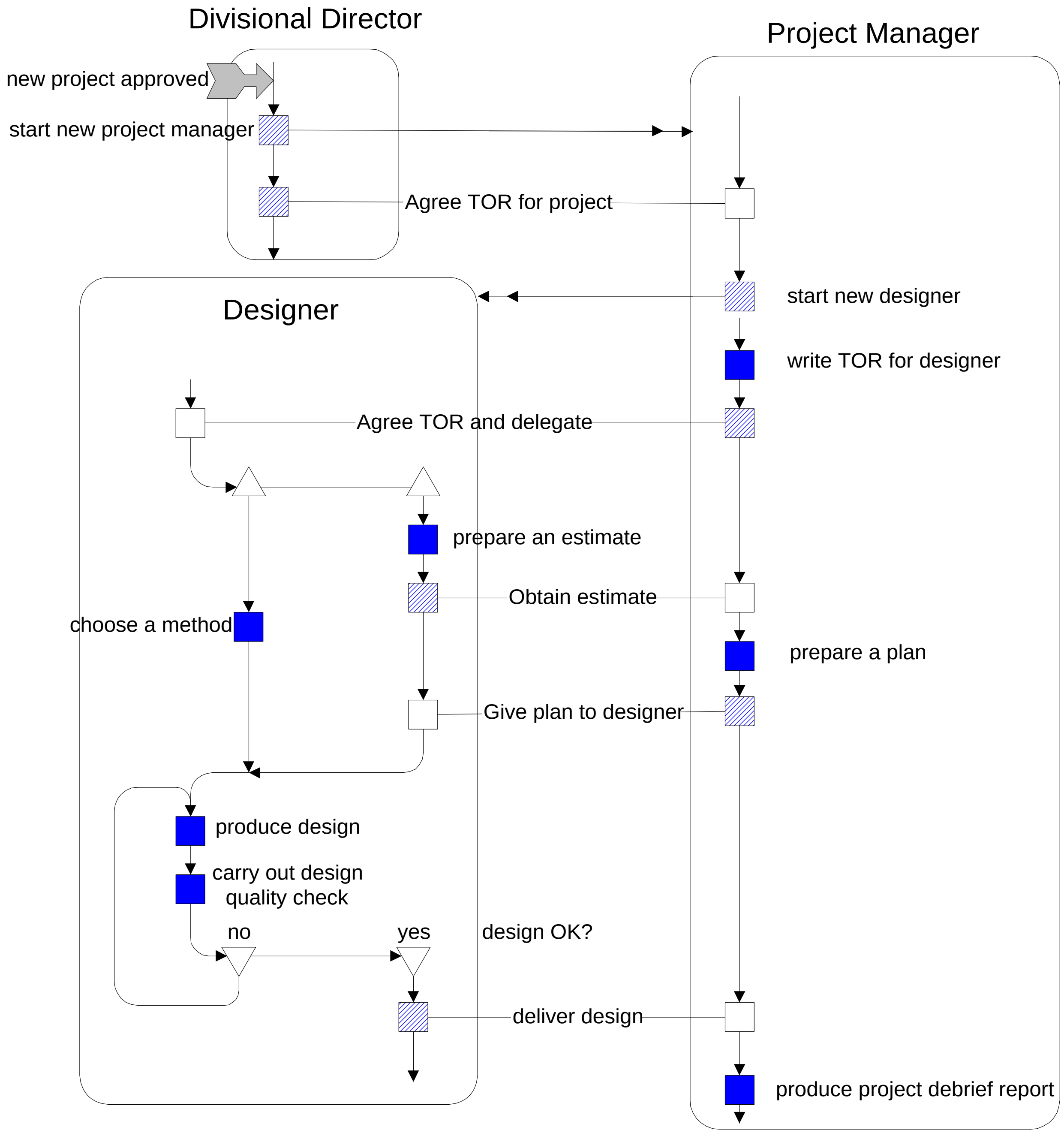
4. The goal was not given

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

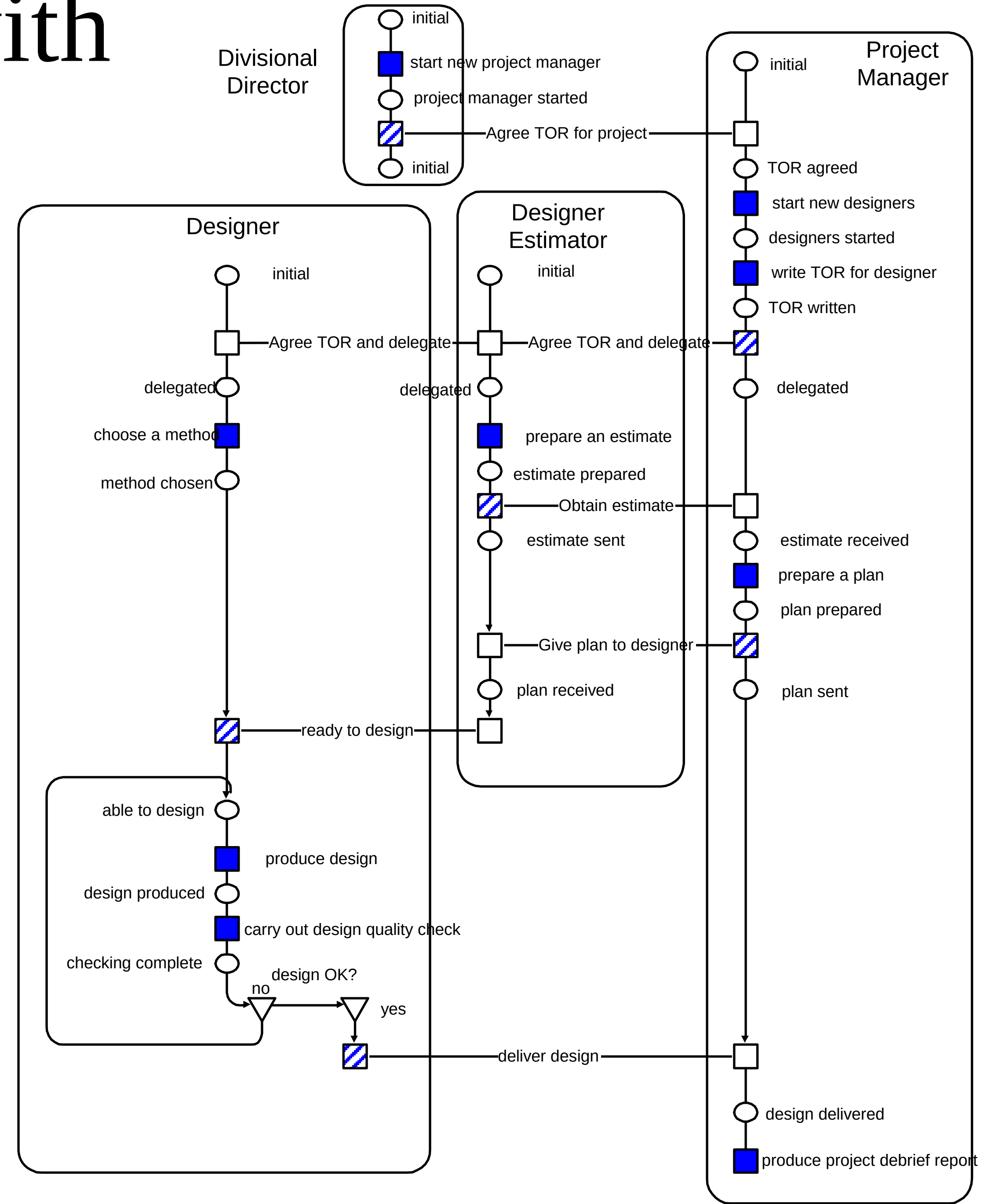
Real agenda

- With many process models (say with RolEnact) users are able to play with behaviour.
  - Lots of arguments about increased understanding, validation etc...
- Wouldn’t that be handy for specifications (as use cases)
  - it’s the old exectuable spec argument again (its so 80s).
- So my analogy is that of RolEnact, which I’ve talked about lots before,
  - (so will show examples - where I’m coming from).

Role Activity Diagram (standard)

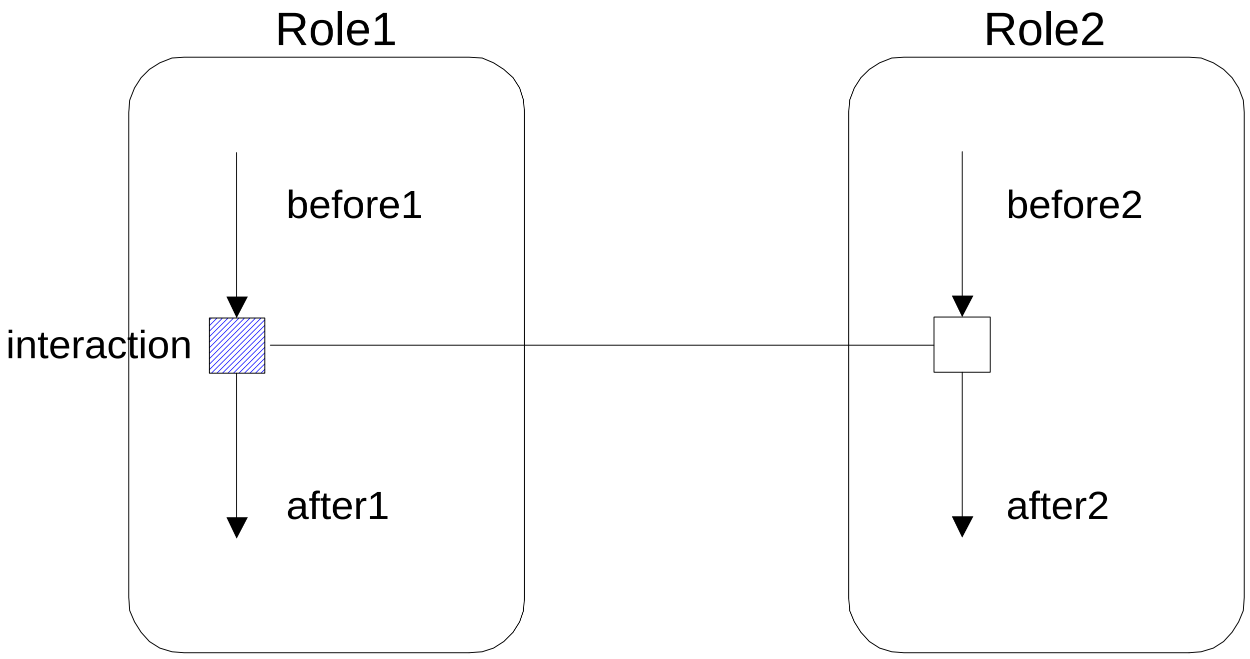


RAD with states



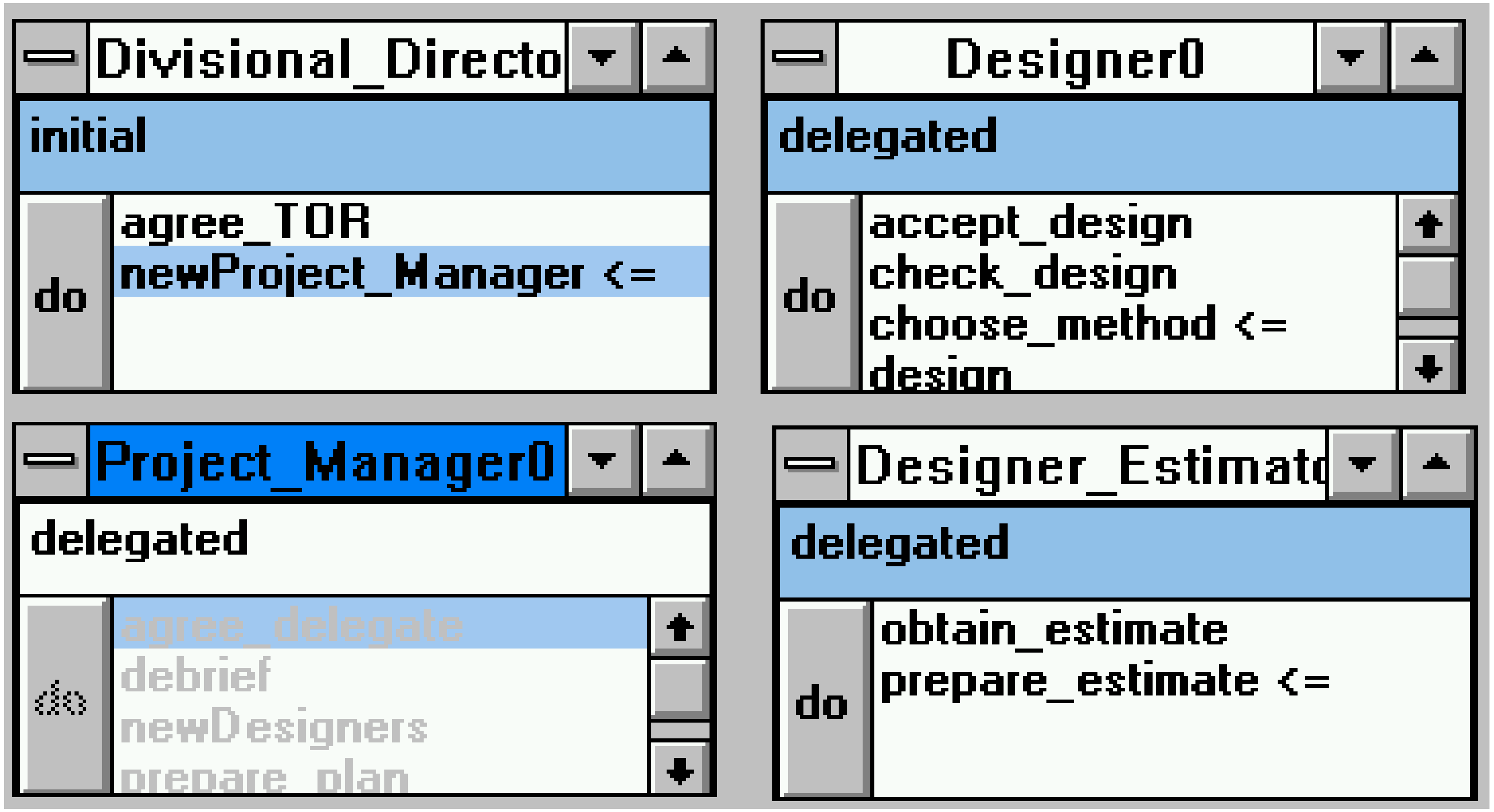
Example RolEnact code

```
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
```

An example enaction?



Divisional_Directo

initial

do

agree_TOR
newProject_Manager <=

Designer0

delegated

do

accept_design
check_design
choose_method <=
design

Project_Manager0

delegated

do

agree_delegate
debrief
newDesigners
prepare_plan

Designer_Estimat

delegated

do

obtain_estimate
prepare_estimate <=

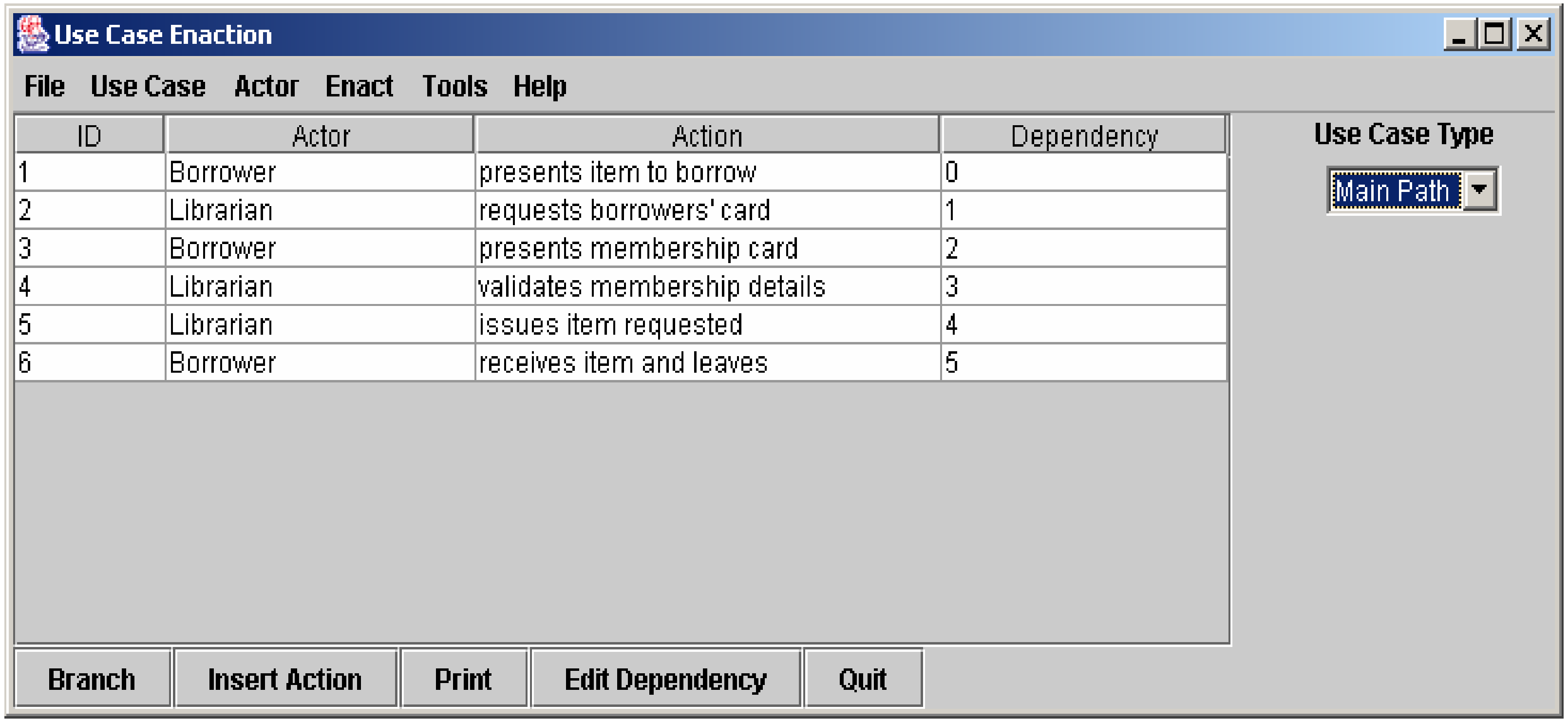
Experiences with Enaction

- Student experience:
  - Can write RolEnact equivalent to use case description and validate with enaction
    - helps tease out issues..
  - Role Activity Diagrams, RolEnact, Use Cases as part of a method
    - strong combination as a requirements validation mechanism
- Industrial experience:
  - Programming to enact each Use Case seen an unwelcome overhead. Not feasible for industrial application.

Use Case Enaction Tool(s)

- The prototype includes:
  - Pre- and post-conditions for each Use Case
  - Text editing capability for standalone Use Case
  - Default dependency capability and Branch dependencies (alternative / exceptions)
  - Enaction of the Use Case
  - Scenario generation of the path selected during enaction
  - Grammar enforcement capability

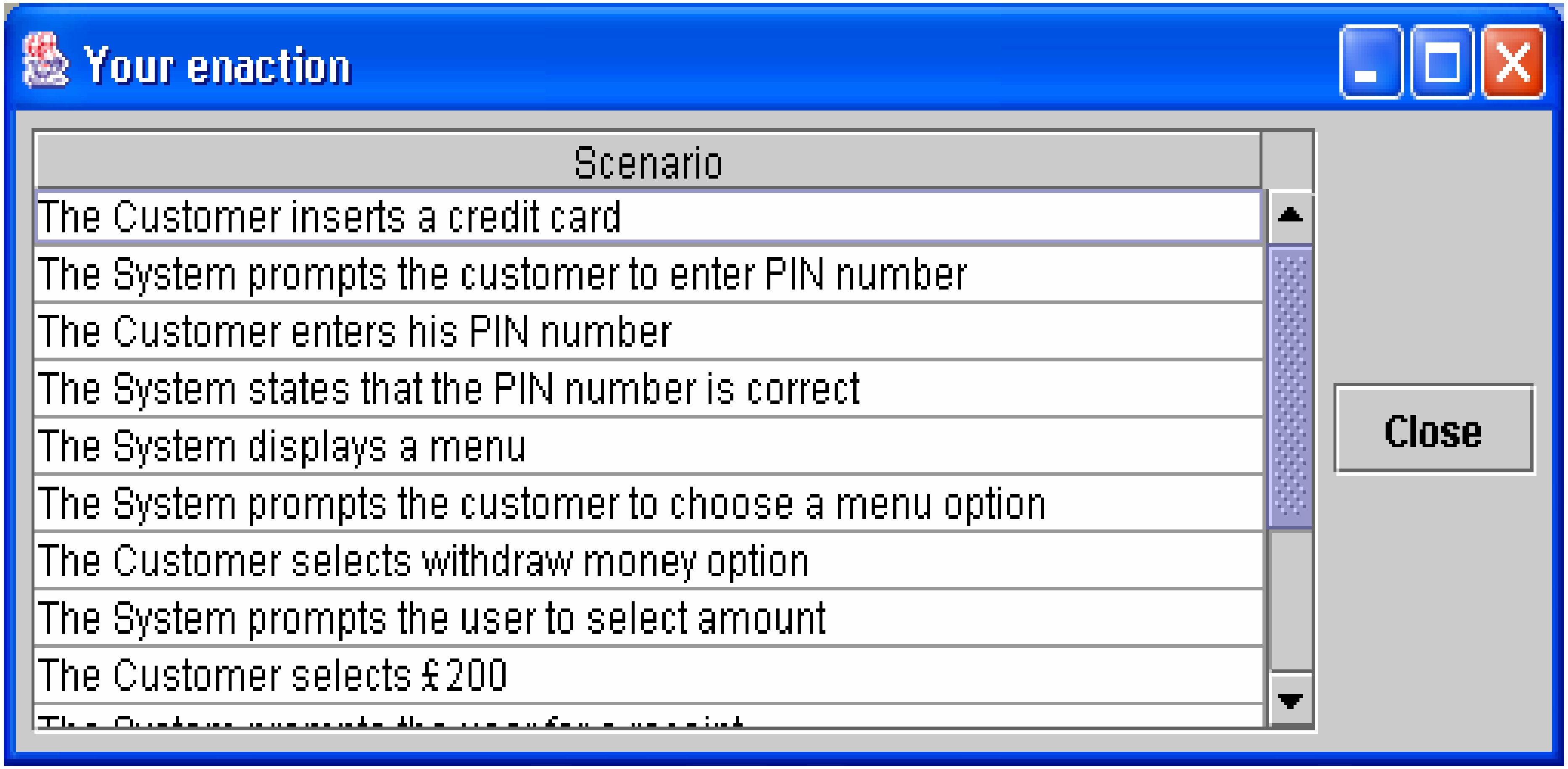
Previous version: Use Case Editor



Use Case Type: Main Path

ID	Actor	Action	Dependency
1	Borrower	presents item to borrow	0
2	Librarian	requests borrowers' card	1
3	Borrower	presents membership card	2
4	Librarian	validates membership details	3
5	Librarian	issues item requested	4
6	Borrower	receives item and leaves	5

Example Enaction



Scenario
The Customer inserts a credit card
The System prompts the customer to enter PIN number
The Customer enters his PIN number
The System states that the PIN number is correct
The System displays a menu
The System prompts the customer to choose a menu option
The Customer selects withdraw money option
The System prompts the user to select amount
The Customer selects £200

Problems

- Abbreviated dependency mechanism only makes sense at system level / single actor.
- Strength (point) of enaction lost.
- Not helpful for considering AND, where two precondition on two or more actors.
  - Note AND implicit in an interaction.
- Currently revising interface.

Revised interface plan

Me				Actor 2		
Actor name	**Event**	**pre**	**post**	**Actor name**	**pre**	**post**
Keith	gives pen	has pen	no pen	Mathenge	no pen	has pen
Mathenge	gives pen	has pen	no pen	Keith	no pen	has pen
Me				**Actor 2**		
Actor name	**Event**	**pre**	**post**	**Actor name**	**pre**	**post**
Driver	drives to ticket machine	initial	at machine			
Driver	presses the ticket button	at machine	ticket requested	Ticket Machine	initial	ticket requested
Ticket Machine	dispenses ticket	ticket requested	ticket dispensed			
Driver	takes ticket	ticket requested	ticket taken	Ticket Machine	ticket dispensed	ticket taken

- See example?

Also for future Construction

- Levels of Usage
  - Advanced usage (detailed dependency selection) versus basic user.
- Multiple use cases
  - Depicting dependencies and enaction across use cases (via include and extend relationships)
- Further flexibility in editing the description
  - e.g., ability to re-order events simply.

Advantages of Tool Support: well here’s hoping

- Use Cases dependency examination offers insights into:
  - the problem domain, the requirements and later in subsequent design
  - and is important to requirements validation.
- Enaction thus provides this dependency scrutiny at ‘minimum’ effort for clients.

Some Issues for tool support

- Does the increased capability offered by dependencies enhance or overcomplicate descriptions?
- Will the inclusion of use case writing guidelines restrict the flexibility offered by enaction?
- 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?