Enactable (Role) Models

- Continue illustration of process ideas by reference to role based models.
- Diagrams allow understanding - but often sacrifice rigour.
- Rigour can be added with..
  - Heuristics, measures, formality, enaction (simulation).
- Initially examine the idea of enaction.

Faith, Maths, Validation and Debugging

- Formal: Rigour: Need checking by experts
- Pragmatic: Typically diagrams:
  - Sacrifice rigour for understandability.
  - Users validate. Suffer from multiple interpretation
- Enactable. Visual. Try it out.
- Combinations.
  - Understandability and rigour.
  - Flexibility and familiarity.
  - Separation of concerns.

RolEnact

- A language for strategic (process) modelling.
- Formal semantics; based upon a condition-action paradigm.
- Primitives match those of role-based models (such as RADs).
- Processes described in terms of roles, the states of these roles, and the activities or events in which each role may take part.

RolEnact Role Instances

- An instance of a role has state, it may have data attached to it, and it may move to its next state through an activity. This activity may be in isolation (an action) or may involve changing the state of another role or roles (an interaction or a selection).
- Note the similarity between Roles (like classes) and instances of roles (like objects).

Advantages of RolEnact

- Brings together condition-action and role based paradigms.
- May be executed on a computer providing a simple Windows-based interface
- These enactable models are used by two main classes of users, modellers and representatives of the client.

RolEnact Users

- Modellers: who produce and experiment with the models
  - understand the process description, discover problems and analyse alternatives.
- Representatives of the client organisation, who interact with the models by taking the parts of users.
  - Validate models, experiment with scenarios and provide a vehicle for discussion.

Moving Towards Enaction

- RADs describe types, they do not describe the synchronisation of instances of the roles.
  - To run simulations, assumptions need to be made about the states of instances of roles.
- RolEnact uses the fact that roles (and hence instances of roles) are viewed as acting in parallel.
  - Parallel threads represented as separate roles.

Representing Parallel

- Parallel threads as separate roles, joining again via an interaction.
- Provides a consistent mapping.
- Decomposes business such that parallel threads could now be assigned to different actors.
- Disadvantages: may be a less representative depiction of the business & may have a greater number of roles.

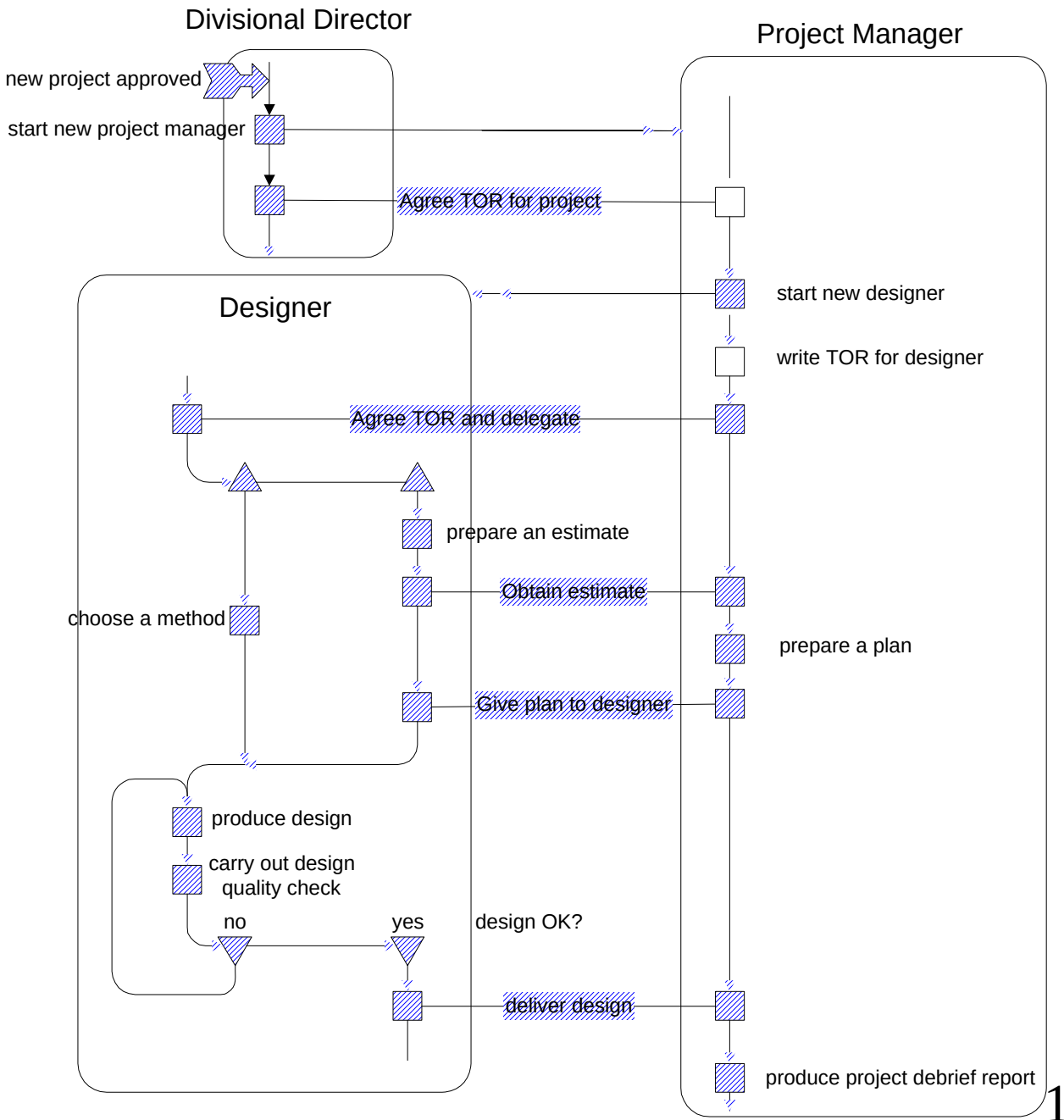
Building Blocks

- All RolEnact models can be made up of four basic types of behaviour; action, interaction, selection, and creation.
- These behaviours allow instances of roles to move from existing states into new states, to communicate with each other, to choose and then interact with other role instances, and to create new role instances.

Relationship to RADs

- Actions & interactions correspond to the same RAD constructs.
- Selections are interactions between roles which have not previously communicated.
- Creation allows roles to create other role instances and to set up identifiers for future communication.

Example: Designer

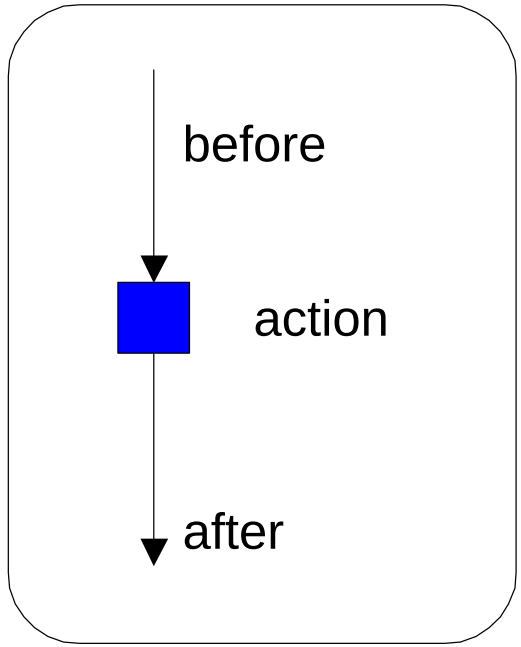


Action

Action Role.Action

Me(before → after)

End



Action Project_Manager.prepare_a_plan

Me(estimate_received → plan_prepared)

End

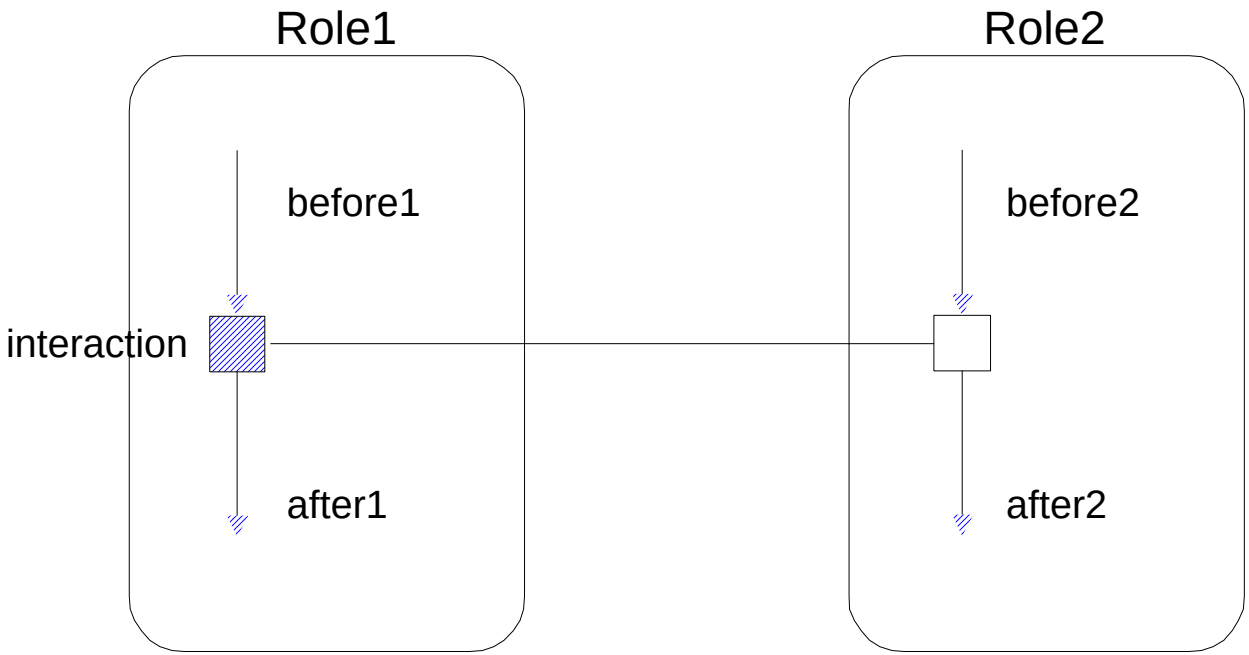
Interaction

Interaction Role1.Interaction

Me(before1 → after1)

Role2(before2 → after2)

End



Selection

Selection Role1.Selection

Me(before1 → after1)

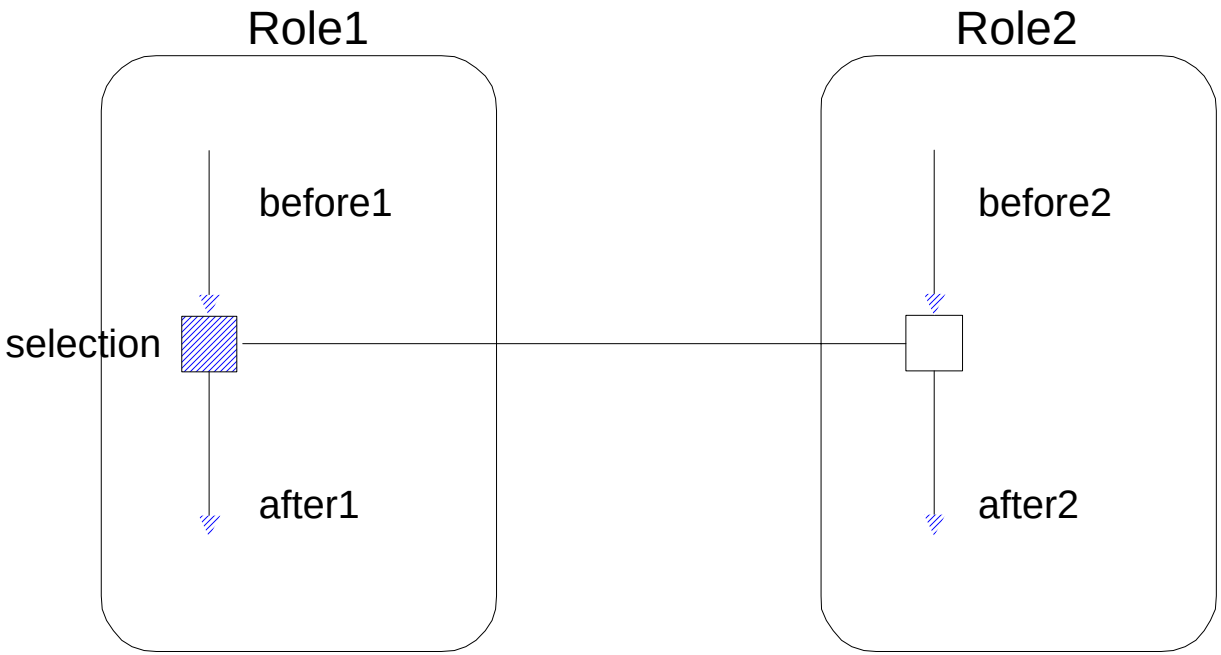
Role2(before2 → after2)

End

Automatically creates:

Me.Role2:=r,

r.Role1:=Me



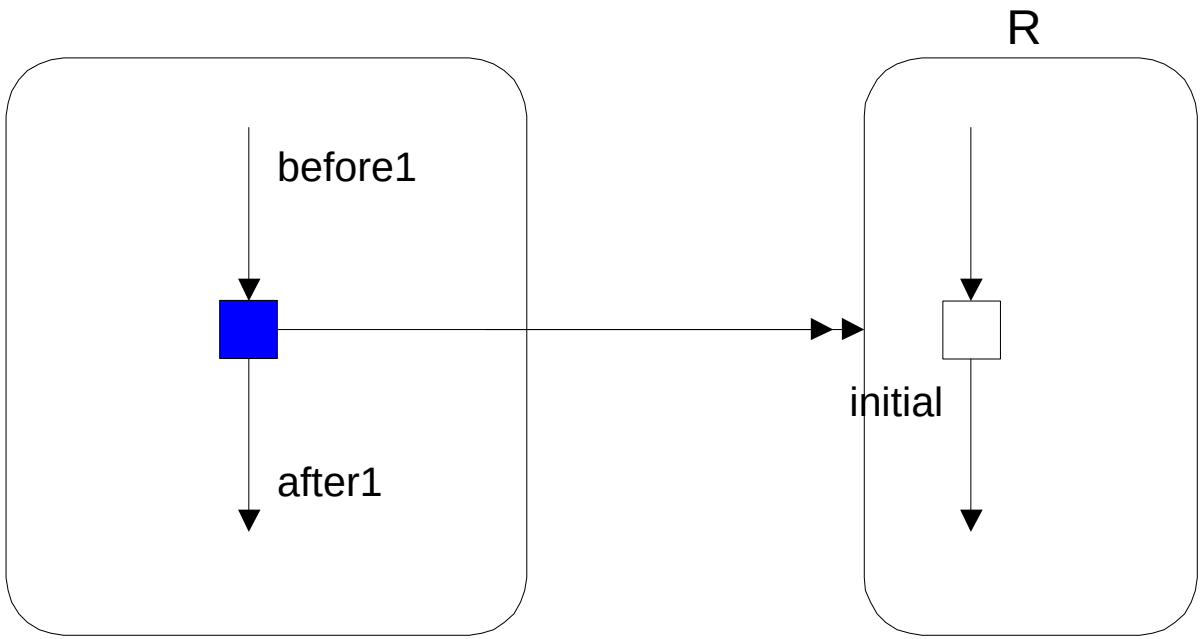
Creation

Create Role1.Create

Me(before1 → after1)

new Role2

End



A Role: Director

Create Divisional_Director.newProject_Manager

me(initial $\rightarrow$ manager_started)

new Project_Manager

End

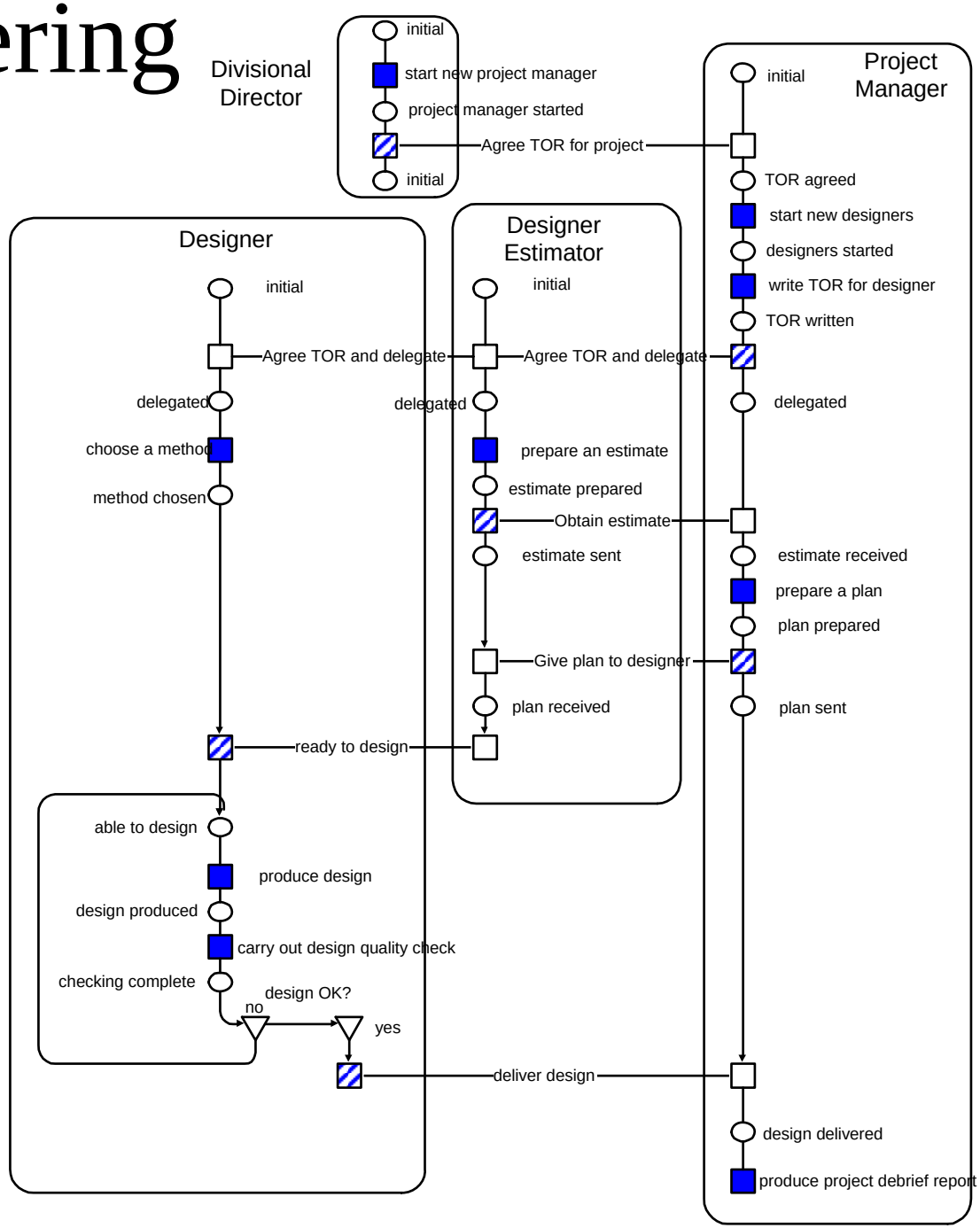
Interaction Divisional_Director.agree_TOR

me(manager_started $\rightarrow$ initial)

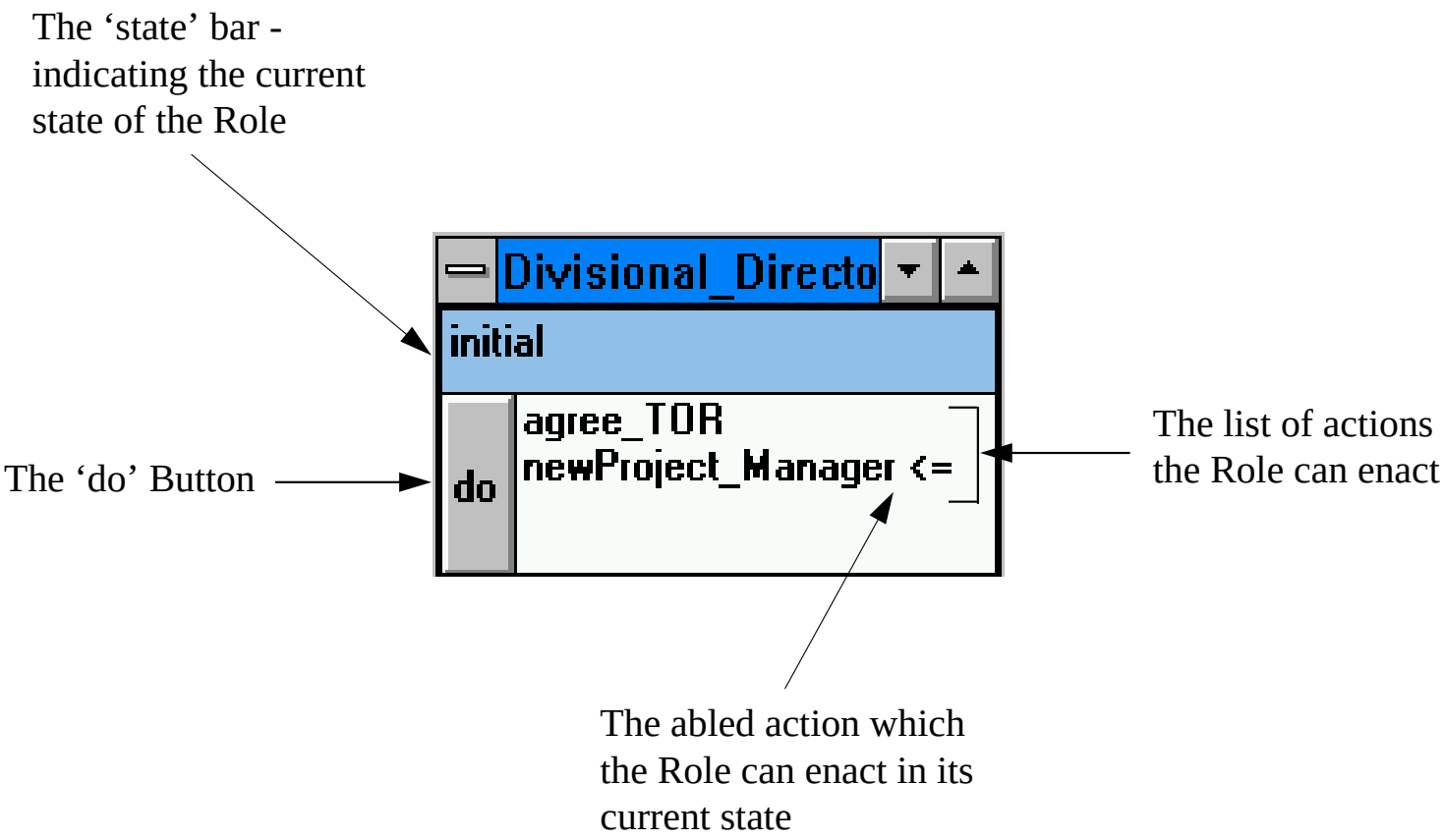
Project_Manager(initial $\rightarrow$ agreed_TOR)

End

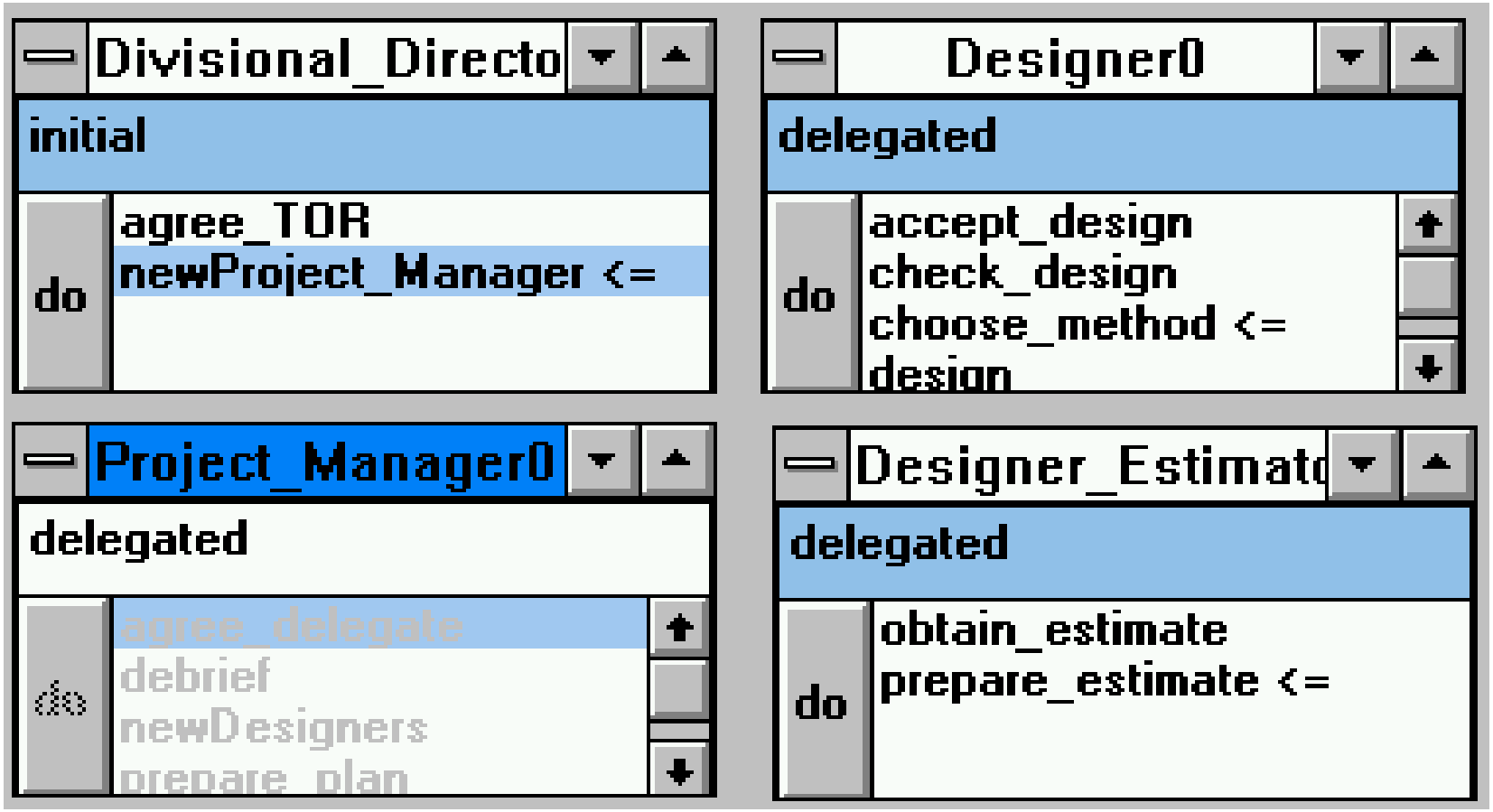
Considering States



RolEnact Windows Interface



RolEnact for Designer



Project Manager

Create Project_Manager.newDesigners
me(agreed_TOR → designers_started)
new Designer
new Designer_Estimator
End

Action Project_Manager.write_TOR
me(designers_started → TOR_written)
End

Interaction Project_Manager.agree_delegate
me(TOR_written → delegated)
Designer(initial → delegated)
Designer_Estimator(initial → delegated)
End

Action Project_Manager.Prepare_plan
me(estimate_received → plan_prepared)
End

Interaction Project_Manager.send_plan
me(plan_prepared → plan_sent)
Designer_Estimator(sent_estimate → received_plan)
End

Action Project_Manager.debrief
me(design_received → project_completed)
End

Designer

Action Designer.choose_method

me(delegated → method_chosen)

End

Interaction Designer.ready_for_design

me(method_chosen → able_to_design)

Project_Manager.Designer_Estimator(received_plan → ended)

End

Action Designer.design

me(able_to_design → design_produced)

End

Action Designer.check_design

me(design_produced → assessing_design)

End

Designer

Action Designer.accept_design

me.(assessing_design → accepted_design)

End

Action Designer.reject_design

me(assessing_design → able_to_design)

End

Interaction Designer.deliver_design

me(accepted_design → design_sent)

Project_Manager(plan_sent → design_received)

Designer_Estimator

Action Designer_Estimator.prepare_estimate

me(delegated → estimated)

End

Interaction Designer_Estimator.obtain_estimate

me(estimated → sent_estimate)

Project_Manager(delegated → estimate_received)

End

Summary: Enactable Models

- Illustrated enaction with very small (proof of concept) enactable notation.
  - Considered mapping from diagram (class description) to enaction (instance description).
  - Illustrated ideas behind client experimentaion with business model.
- Remember rigour can also be added with..
  - Heuristics, measures, or formality.

.new and .rol

- In reality a further conversion happens before ENACT uses the file.
  - Translator translates the .new file to a .rol file
  - (the real and more poweful RolEnact language).
- Often easier to write a .rol file.
  - Consider mapping (macros) .new to .rol
- Examine the main constructs again
  - and note subtle additions in amul.rol.
- Examine language description (.act) file.

Action

Action ROLE.ACTION

    Me(BEFORE -> AFTER)

End

ROLE.addEvent('ACTION).

    ROLE.ACTIONReady():=self.inState('BEFORE).

    ROLE.ACTIONDoit():=self.setState('AFTER).

Interaction

```
Interaction ROLE1.ACTION
    Me(BEFORE1 -> AFTER1)
    ROLE2(BEFORE2 -> AFTER2)
End

ROLE1.addEvent('ACTION).
    ROLE1.ACTIONReady():=
        self.inState('BEFORE1) and
        self.ROLE2.inState('BEFORE2).
    ROLE1.ACTIONDoit():=
        (self.setState('AFTER1);
        self.ROLE2.setState('AFTER2)).
```

Selection

```
Selection ROLE1.ACTION
    Me(BEFORE1 -> AFTER1)
    ROLE2(BEFORE2 -> AFTER2)
End

ROLE1.addEvent('ACTION).
    ROLE1.ACTIONReady():=
        self.inState('BEFORE1) and
        ROLE2.exists('BEFORE2).
    ROLE1.ACTIONDoit():=
        (aRole.choose('BEFORE2);
        self.setState('AFTER1);
        aRole.setState('AFTER2);
        self.ROLE2:=aRole;
        aRole.ROLE1:=self).
```

Creation

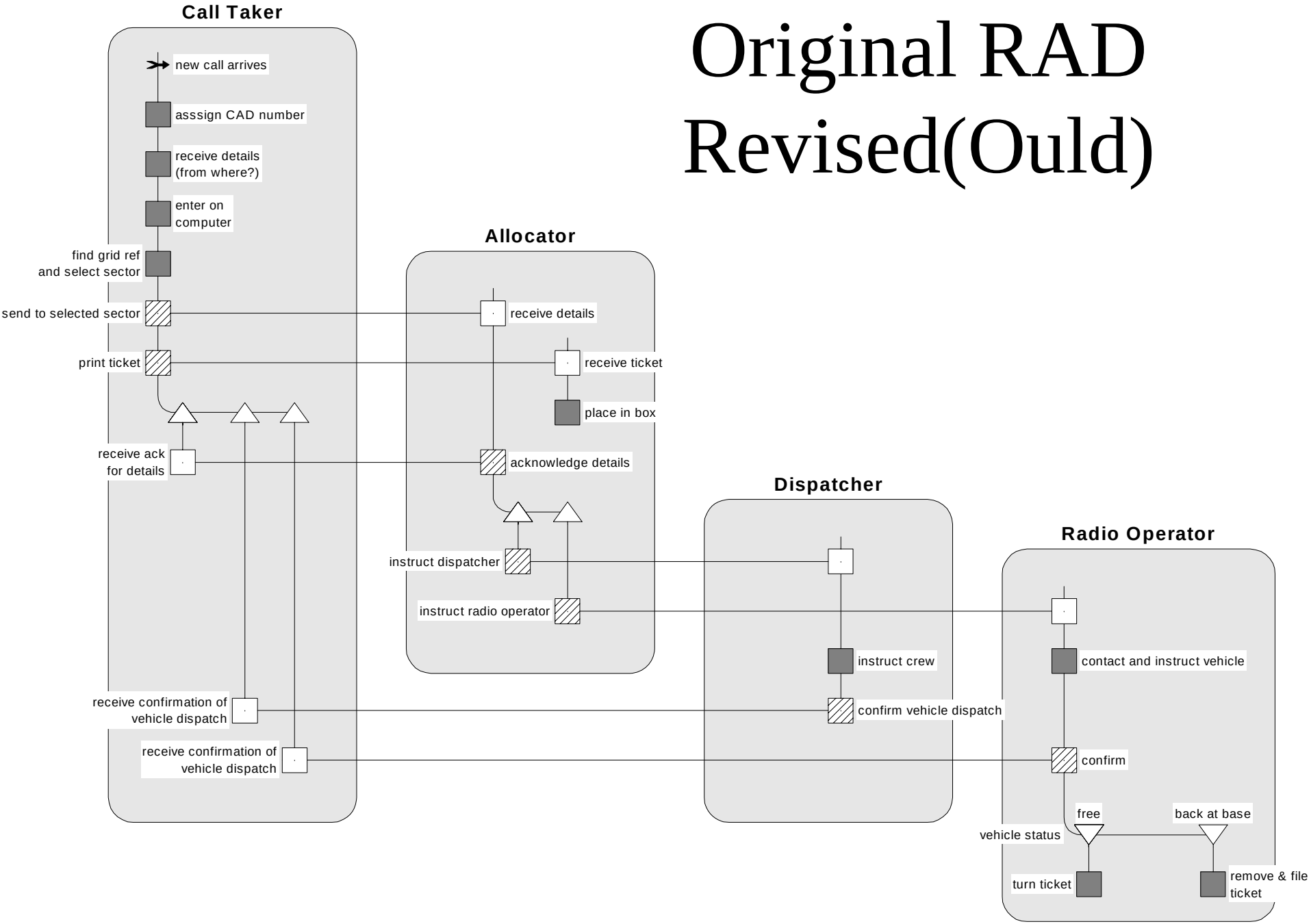
```
Create ROLE1.ACTION
    Me(BEFORE1 -> AFTER1)
    new ROLE2
End

ROLE1.addEvent('ACTION).
    ROLE1.ACTIONReady():=
        self.inState('BEFORE1).
    ROLE1.ACTIONDoit():=
        (self.setState('AFTER1);
        self.ROLE2:=ROLE2.create();
        self.ROLE2.ROLE1:=self).
```

Running Enact

- Your mission is...
- Find appropriate version of RolEnact
  - Note some graphical tools (REPI)
  - Translator files (.new to .rol)
  - Basic RolEnact (RolEnact.exe)
- Download / install to a suitable (e.g., home) machine.
- Play and learn. You will need to use this later in the course.

Original RAD Revised(Ould)



RAD from Description (Ould)