UNIVERSITY OF Southampton

Language and Tool Support for Class and State Machine Refinement in UML-B

Mar Yah Said, Michael Butler and Colin Snook

(mys05r,mjb,cfs)@ecs.soton.ac.uk

School of Electronic and Computer Science

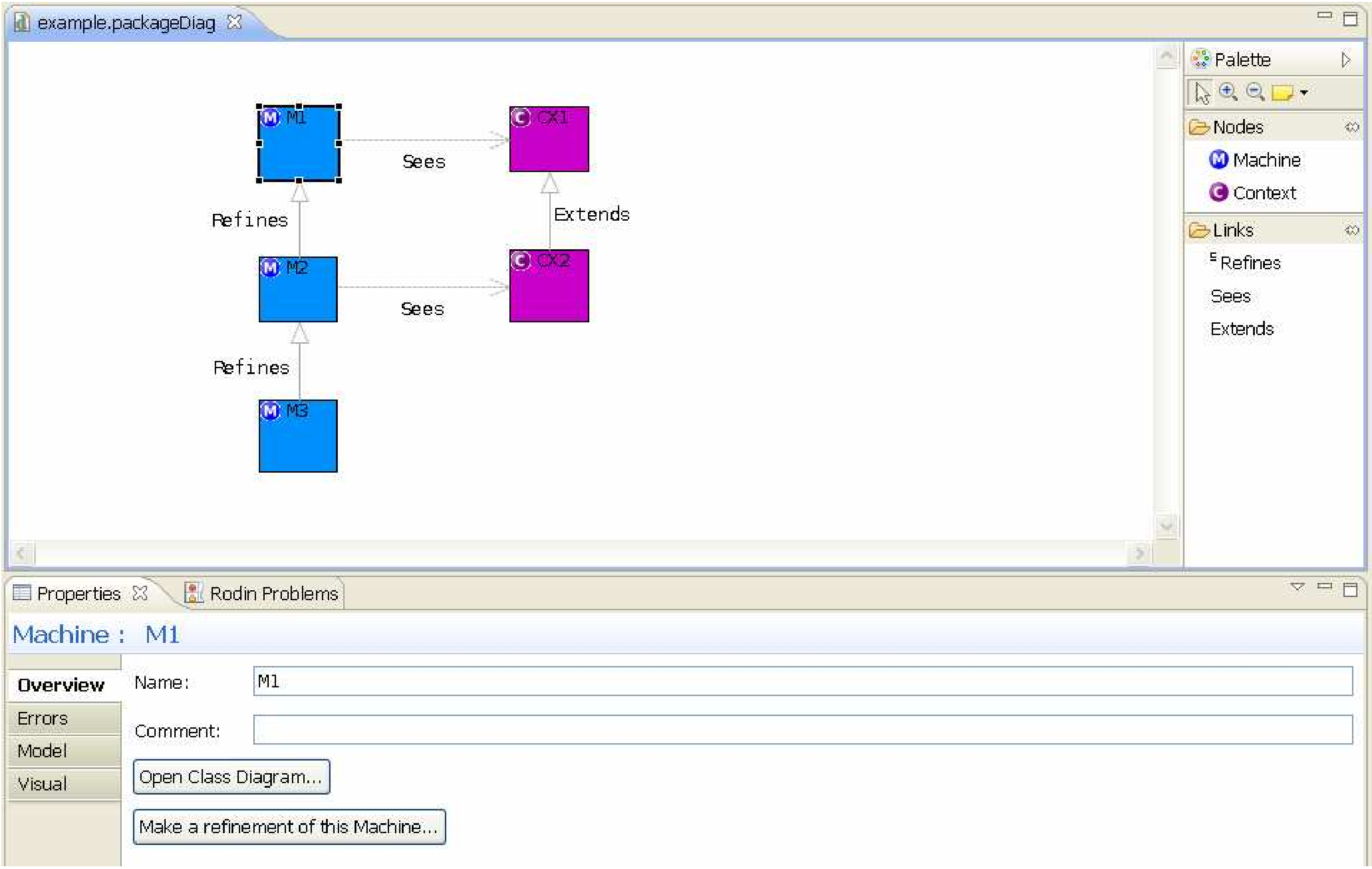
Outline

- Overview of UML-B
- Class refinement
- State machine refinement
- Technique of event movement
- ATM case study
- Conclusion

What is UML-B?

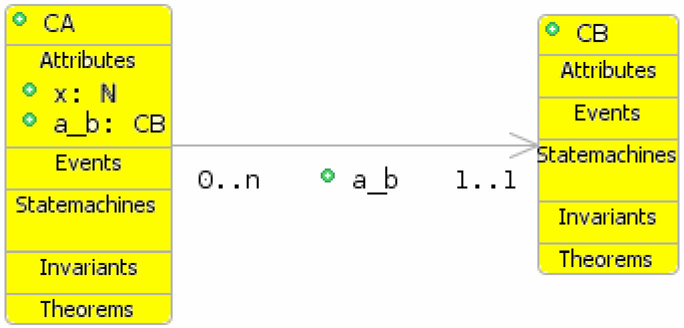
- Is a UML-like graphical front-end to Event-B.
- Supported by UML-B tool – a plug-in extension to Rodin tools.
  - Generates Event-B specification from UML-B diagrams models.
  - Event-B errors are reflected on the UML-B diagrams.
- Four diagrams: package diagram, context diagram, class diagram and state machine diagram

Package Diagram



Class diagram

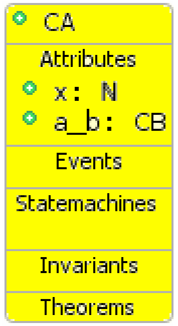
- Classes *CA* and *CB* give rise to the sets *CA_SET* and *CB_SET* in the generated implicit Event-B context.
- In the generated Event-B machine, the classes *CA* and *CB* become variables.
- The attributes *x* and *a_b* *give rise to* variables.



INVARIANTS

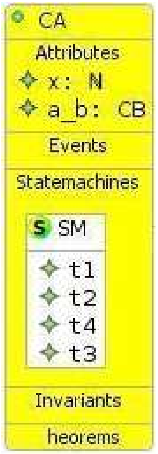
CA.type : $CA \in \mathbb{P}\ (CA\_SET)$
CB.type : $CB \in \mathbb{P}\ (CB\_SET)$
x.type : $x \in CA \rightarrow \mathbb{N}$
a_b.type : $a\_b \in CA \rightarrow CB$

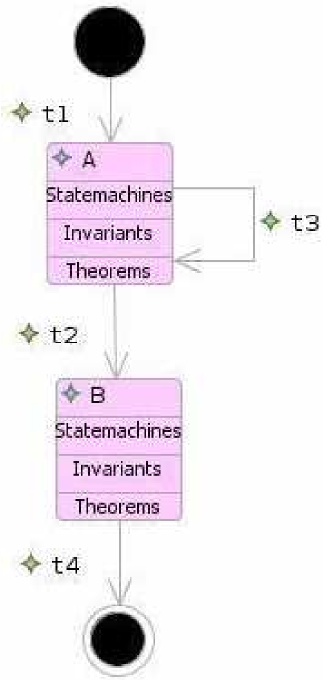
Self name property of a class



- Each class has a *self name* property
- Represents an instance of a class
- Default name is *self*
- Can be change by modeller
- Give rise to a parameter of an event in the Event-B machine

State Machine of the class *CA*





- Class CA has a state machine *SM*
- Disjoint sets representation
  - A disjoint sets of CA are introduced as variables:

$$A \in \mathbb{P}(CA)$$
$$B \in \mathbb{P}(CA)$$
$$A \cap B = \{\ \}$$

  - Variable A represents the set of instances of CA that are in the state A.

- State function representation
  - A variable SM is introduced representing a function mapping CA to an enumerated set of states, *SM_STATES*:

$$SM\_STATES = \{A, B\}$$
$$SM \in CA \rightarrow SM\_STATES$$

  - That is, SM maps each instance of CA to its state.

Generated Event-B Specification (Transitions becomes events)

```
t1    ≙
STATUS
  ordinary
ANY
  self      //   constructed instance of class CA
WHERE
  self.type    :    self ∈ CA_SET \ CA
THEN
  SM_enterState_A    :    A ≔ A ∪ {self}
  CA_constructor    :    CA ≔ CA ∪ {self}
END
```

```
t2    ≙
STATUS
  ordinary
ANY
  self      //   contextual instance of class CA
WHERE
  self.type    :    self ∈ CA
  SM_isin_A    :    self ∈ A
THEN
  SM_enterState_B    :    B ≔ B ∪ {self}
  SM_leaveState_A    :    A ≔ A \ {self}
END
```

```
t3    ≙
STATUS
  ordinary
ANY
  self      //   contextual instance of class CA
WHERE
  self.type    :    self ∈ CA
  SM_isin_A    :    self ∈ A
THEN
  skip
END
```

```
t4    ≙
STATUS
  ordinary
ANY
  self      //   contextual instance of class CA
WHERE
  self.type    :    self ∈ CA
  SM_isin_B    :    self ∈ B
THEN
  SM_leaveState_B    :    B ≔ B \ {self}
  CA_destructor    :    CA ≔ CA \ {self}
  CA.a_b_destructor    :    a_b ≔ {self} ⩤ a_b
  CA.x_destructor    :    x ≔ {self} ⩤ x
END
```

Class Refinement

Notion of Refined Classes (and inherited attributes)

- Motivation: performing refinement in Event-B.
- Reflect the refinement of variables in Event-B.
- Refined class is one that refines a more abstract class.
- Inherited attribute is one that inherits an attribute of the abstract class.

Keep/Drop/New Attributes

Class C

a1

a2

a3

Refined Class C

a1 }
a2 } *inherited*

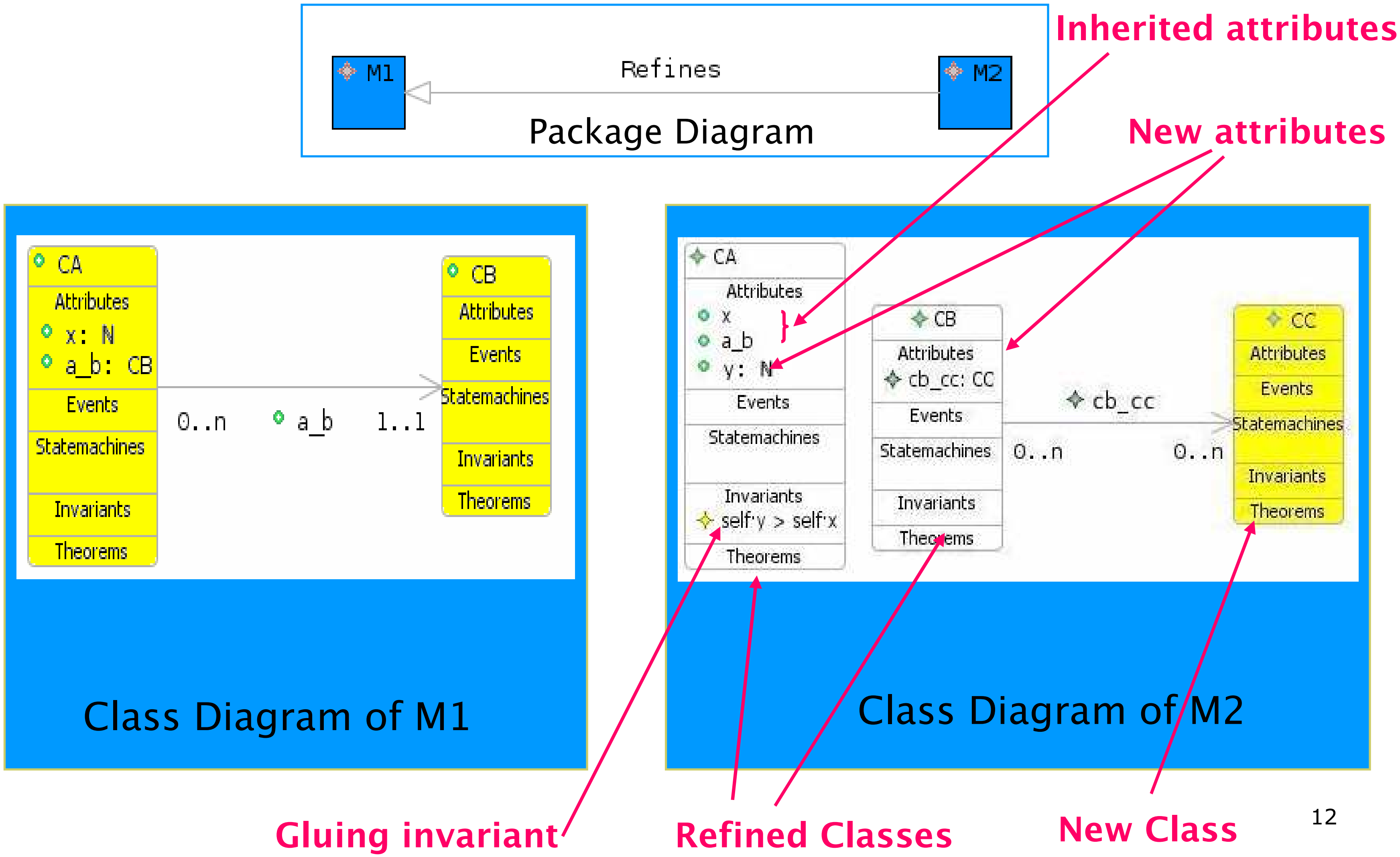
a4 }
a5 } *new*

Similarly for classes: UML-B refinement may contain refined classes, may drop refined classes or/and introduce new classes.

⇒reflects Event-B refinement: keep variables, drop variables, introduce new variables in the refinement.

Refinement of classes in UML-B

(Techniques of adding new classes and new attributes)



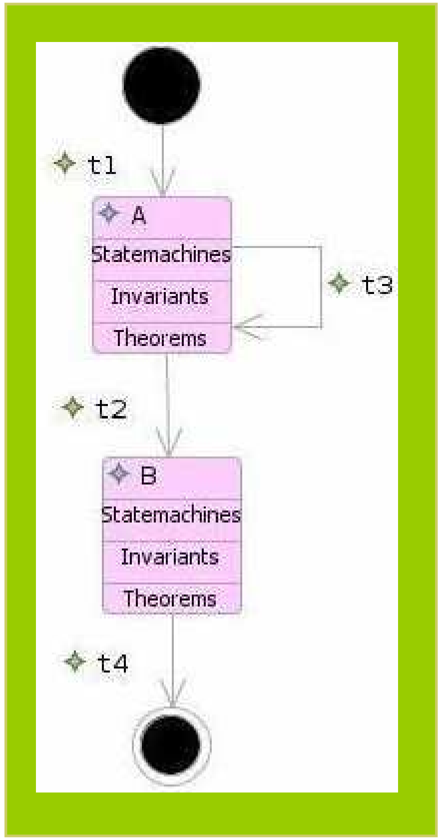
UNIVERSITY OF Southampton

State Machine Refinement

Notion of Refined State Machines (and refined states)

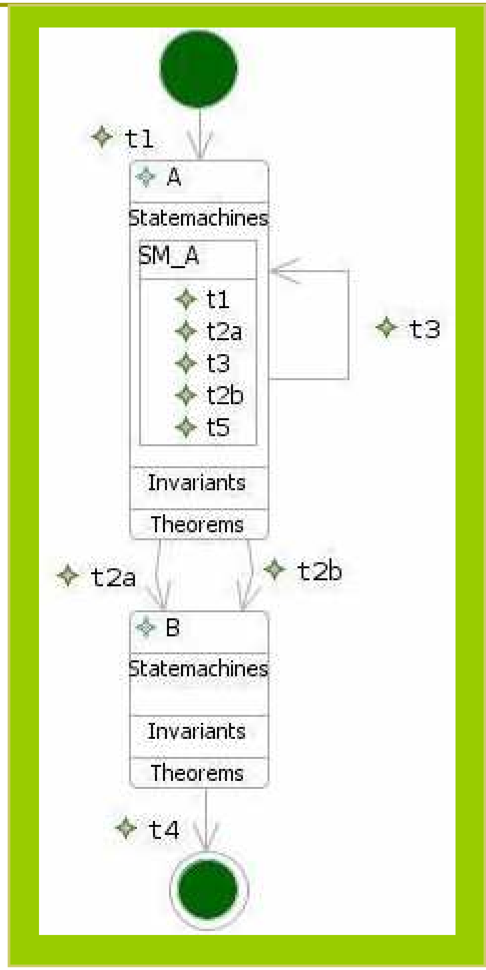
- Motivation: state machine hierarchy in UML-B and refinement in Event-B.
- Refined state machine is one that refines a more abstract state machine.
- Refined state is one that refines a state of the abstract state machine.
- Essential concept : state machines are refined by elaborating an abstract state with nested sub-states.

Refinement of state machines in UML-B



State machine
SM of **M1**

$\sqsubseteq$

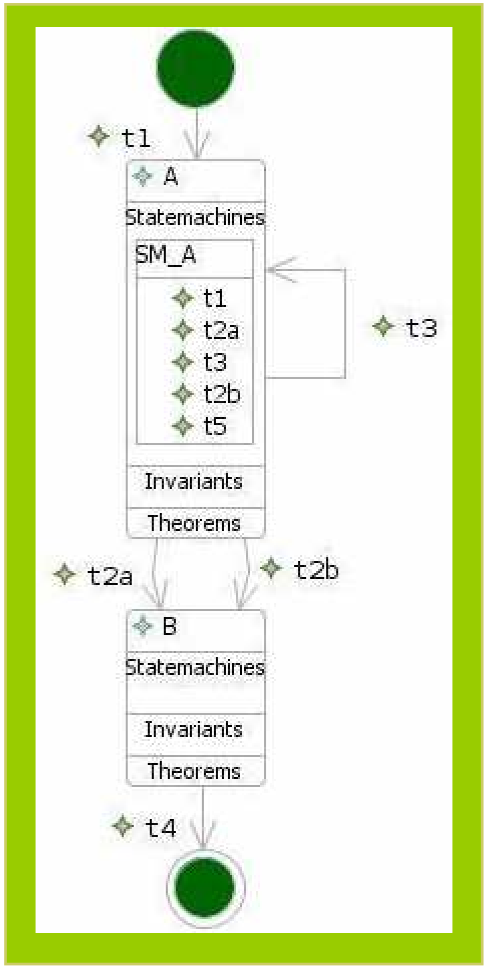


Refined state machine
of **M2** refining *SM* of **M1**

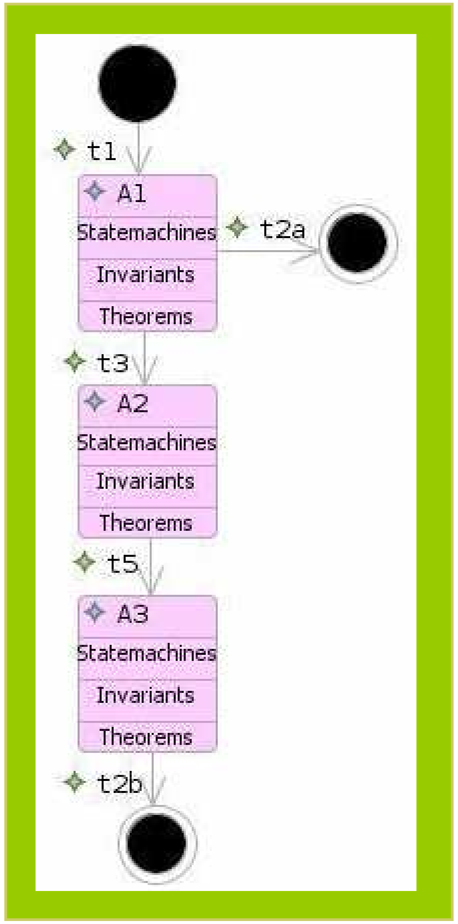
- Transitions *t2* is replaced with transitions *t2a* and *t2b* which refine the abstract transition *t2* of machine M1.
- This refinement of a state machine reflects the refinement in Event-B where many events refine one abstract event.

Nested state machine *SM_A*

(State Elaboration and Transition Elaboration Technique)



Refined state machine
of **M2** refining *SM* of **M1**



State machine
SM_A of **M2**

- The transition *t1* of the nested state machine *SM_A* elaborates the incoming transition *t1* of the refined super-state A.
- The transitions *t2a* and *t2b* of *SM_A* elaborate the outgoing transition *t2a* and *t2b* of the refined super-state A.
- The transition *t3* of the *SM_A* elaborates the self loop transition of the refined super-state A
- Generated gluing invariant:

$$A = A1 \cup A2 \cup A3$$

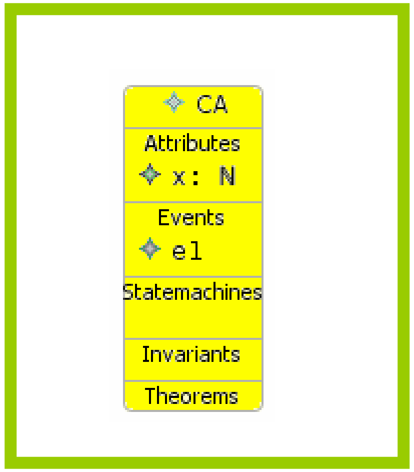
Technique of Event Movement

Technique of Event Movement

- Two methods of moving a class event:
  1. move into a state machine as a transition in a refined class.
  2. move to a new class as a class event or a transition in a state machine.

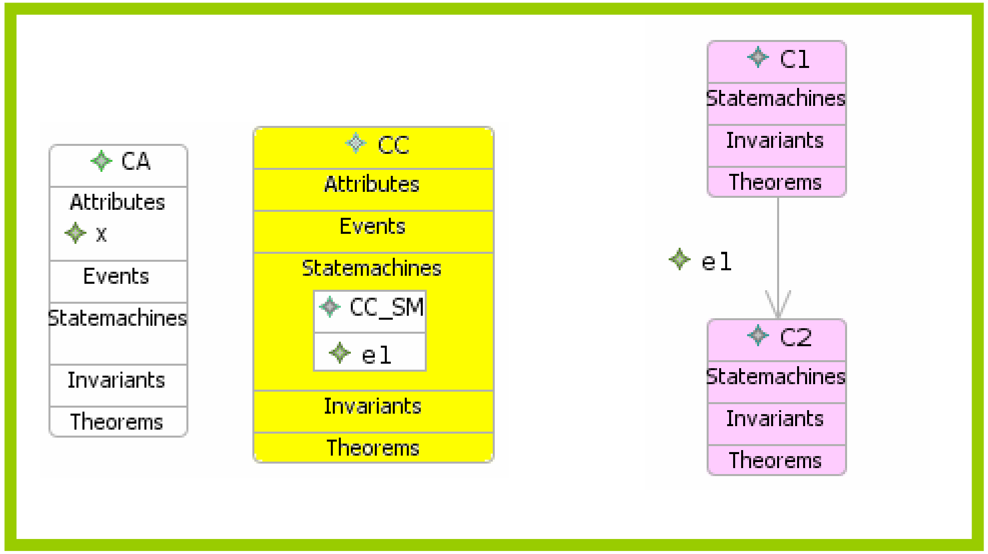
- Method (1) does not need any new UML-B language feature. However, method (2) creates a motivation for the need to be able to change the default *self* name in UML-B.

Technique of Event Movement: Method 2



Abstract machine

$\sqsubseteq$



Refinement machine

- In refinement, the event *e1* is moved to the new class CC as a transition in the state machine *CC_SM*.
- a witness property is defined for the event *e1* :

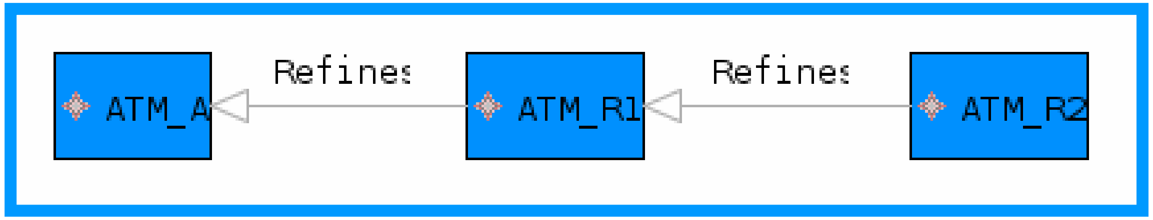
  ca = selfCA

  where *ca* is a parameter of the event *e1* and *selfCA* is an instance of the abstract class *CA*.

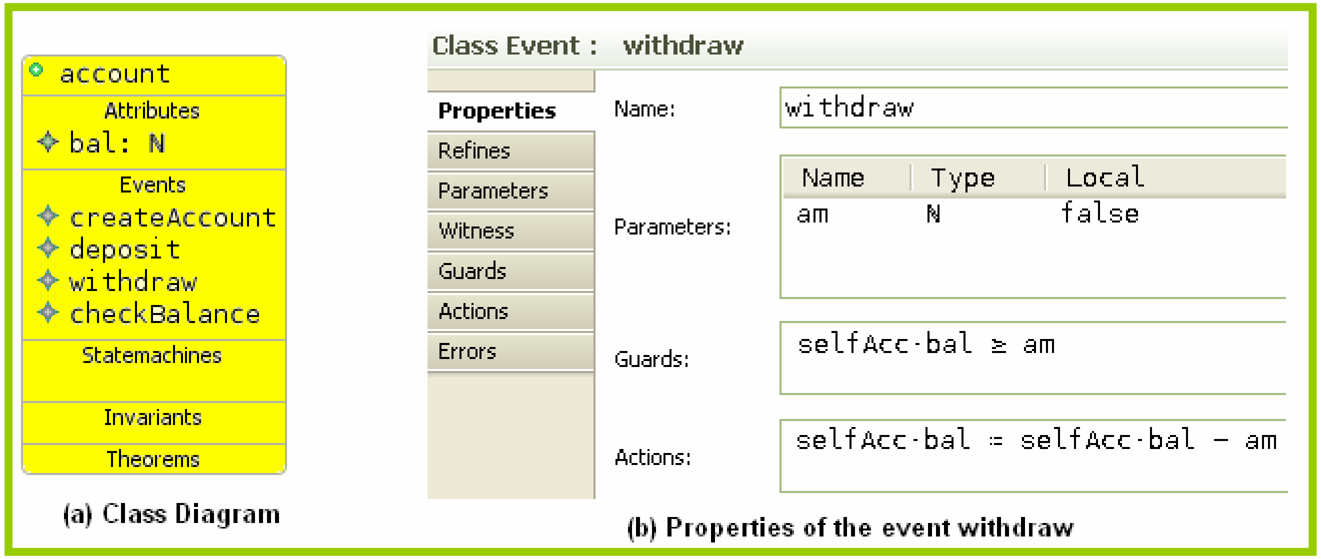
Case Study :
Auto-teller machine (ATM)

Package Diagram of ATM

- There are three machine levels for the ATM UML-B development:
  - Abstract machine (ATM A): Models bank accounts and operations on accounts.
  - First Refinement (ATM R1): Introduces the ATMs, cards and PIN numbers.
  - Second Refinement (ATM R2): Introduces an explicit validation transition for cards and splits withdrawal into a bank transition and an ATM transition.

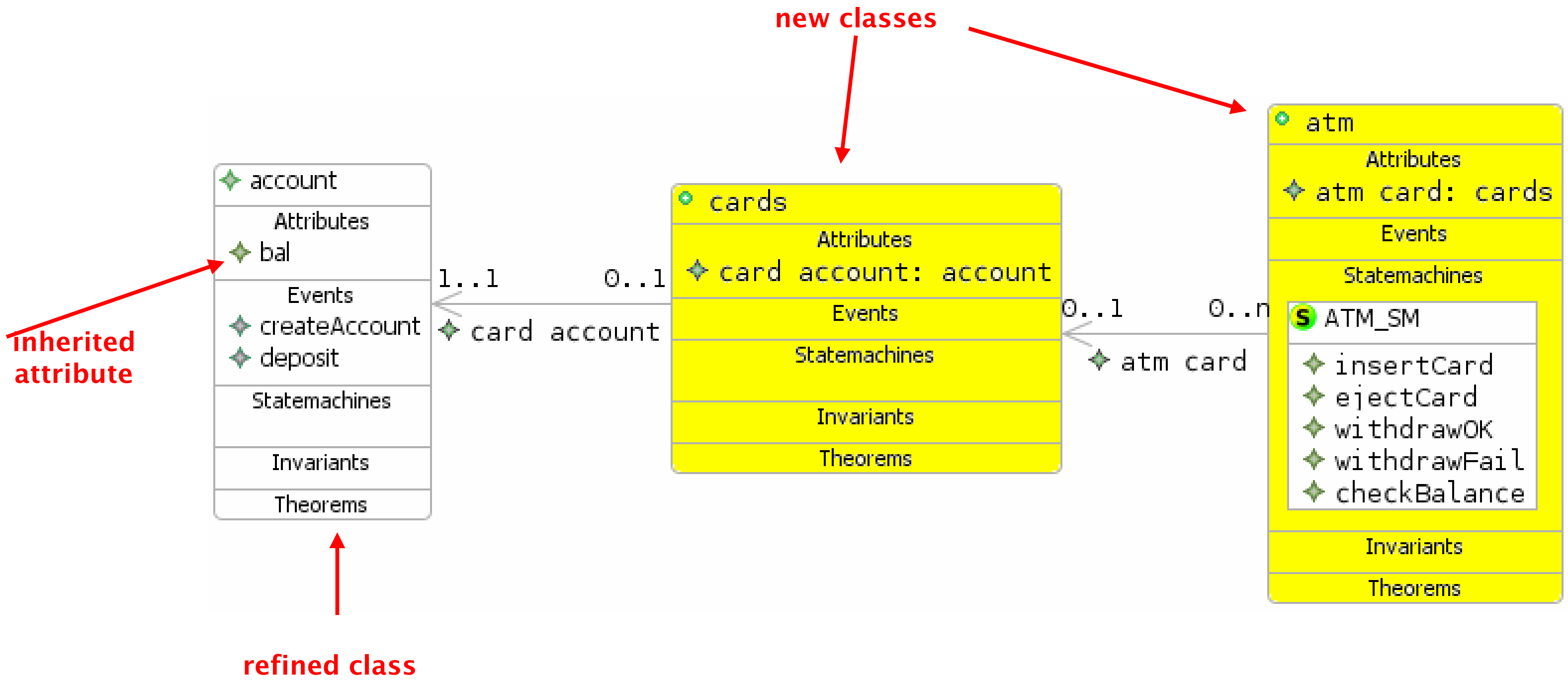


ATM Abstract Machine



- A class *account* (a) has attribute *bal* and four events: *createAccount*, *deposit*, *withdraw* and *checkBalance*.
- The specification of the withdraw event is shown in (b) including parameters, guard and action.
- *selfAcc* is the self name property defined for the class Account.

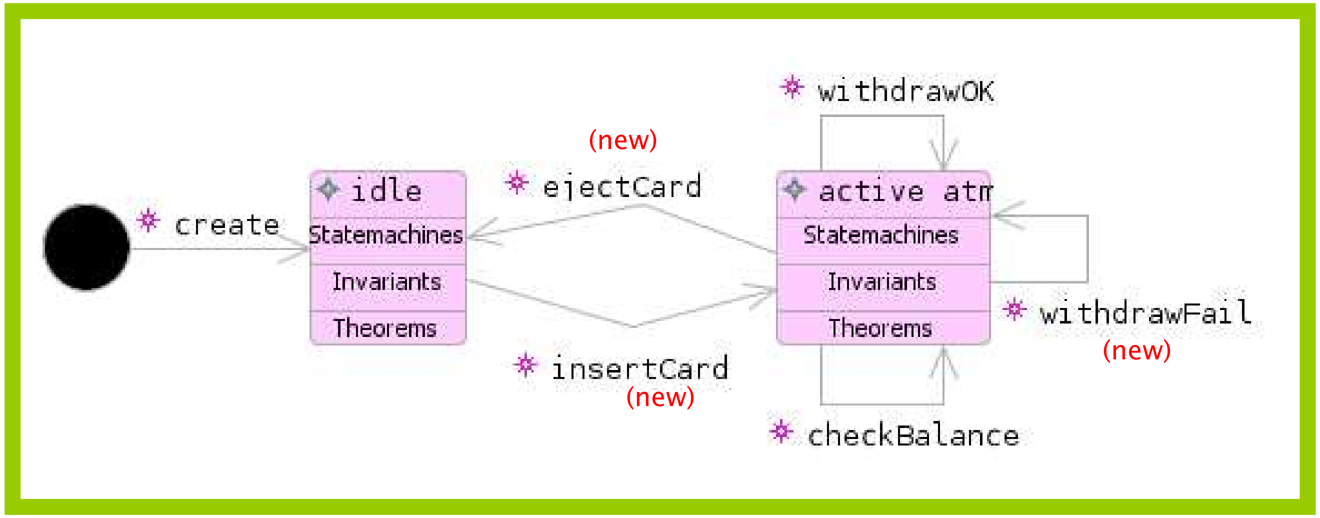
ATM First Refinement : Class Diagram



- The events *withdraw* and *checkBalance* of its abstract class are moved to the new class *atm* in this refinement as transitions in the state machine ATM_SM of the class atm.

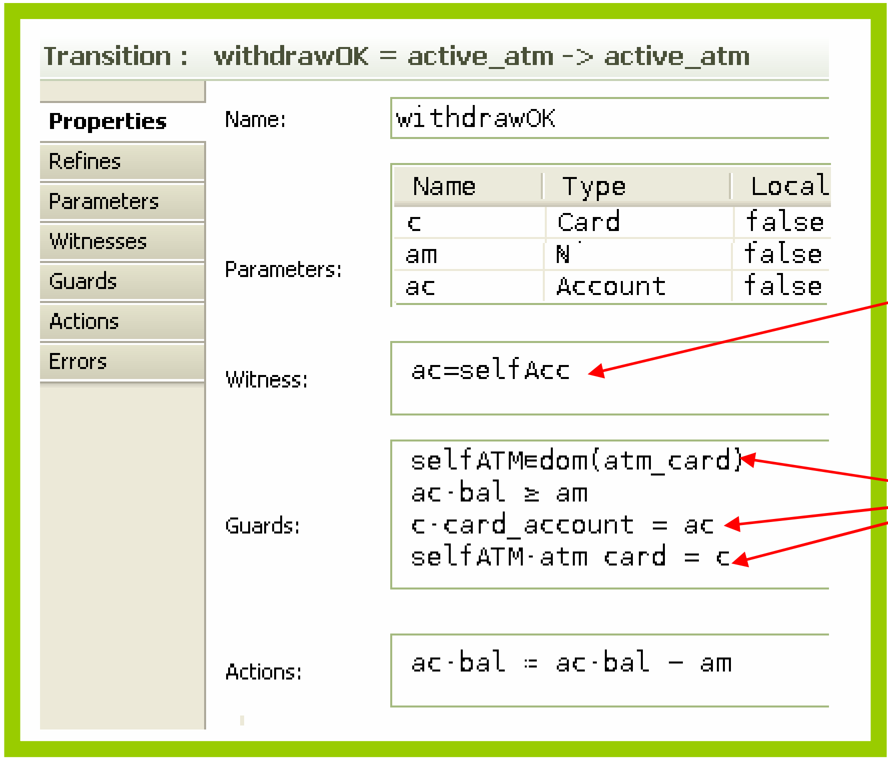
- In the refinement, we specify that the withdrawal takes place via an ATM. At the abstract level it is natural to specify the withdrawal as an event of the Account class while in the refinement it is natural to specify it as an event of the ATM class.

ATM First Refinement : State Machine of ATM Class



- The state machine *ATM_SM* partitions the behaviour of an ATM into an *idle* state, (i.e., not being used/not active) or *active_atm* state (i.e., is being used).
- Three new events: *insertCard*, *ejectCard* and *withdrawFail.*
- Events *withdrawOK* and *checkBalance* refine the abstract events.

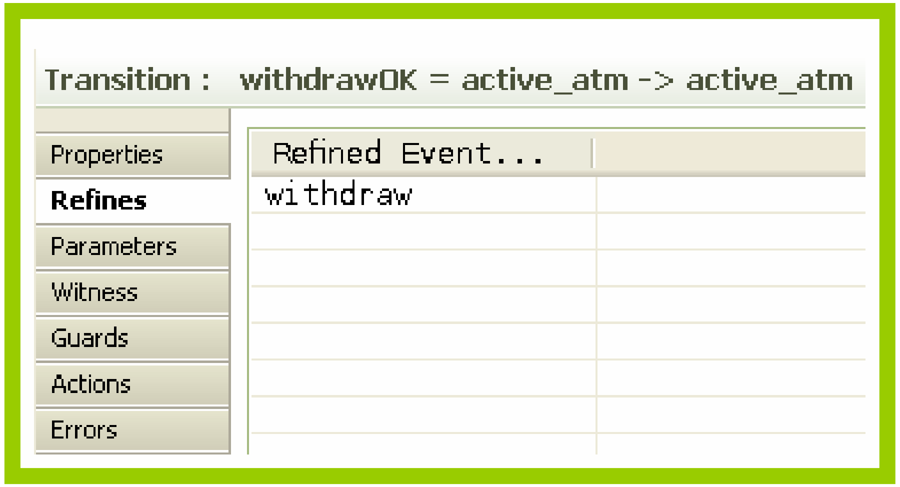
ATM First Refinement: Properties of *withdrawOK* event



A witness specifying that *ac* represents the *selfAcc* of the abstract event

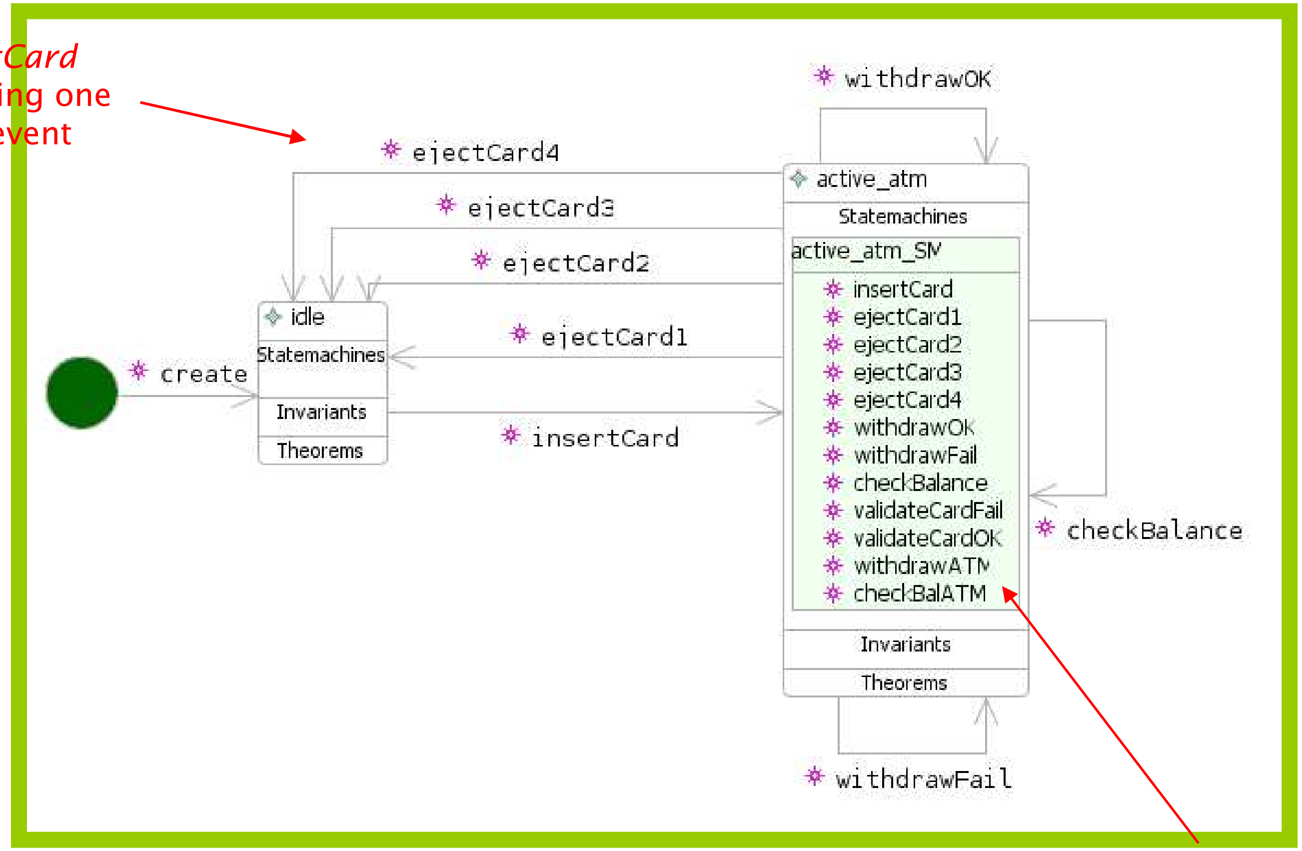
The guards are strengthened

Refines properties of *withdrawOK* event



ATM Second Refinement: Refined state machine *ATM_SM*

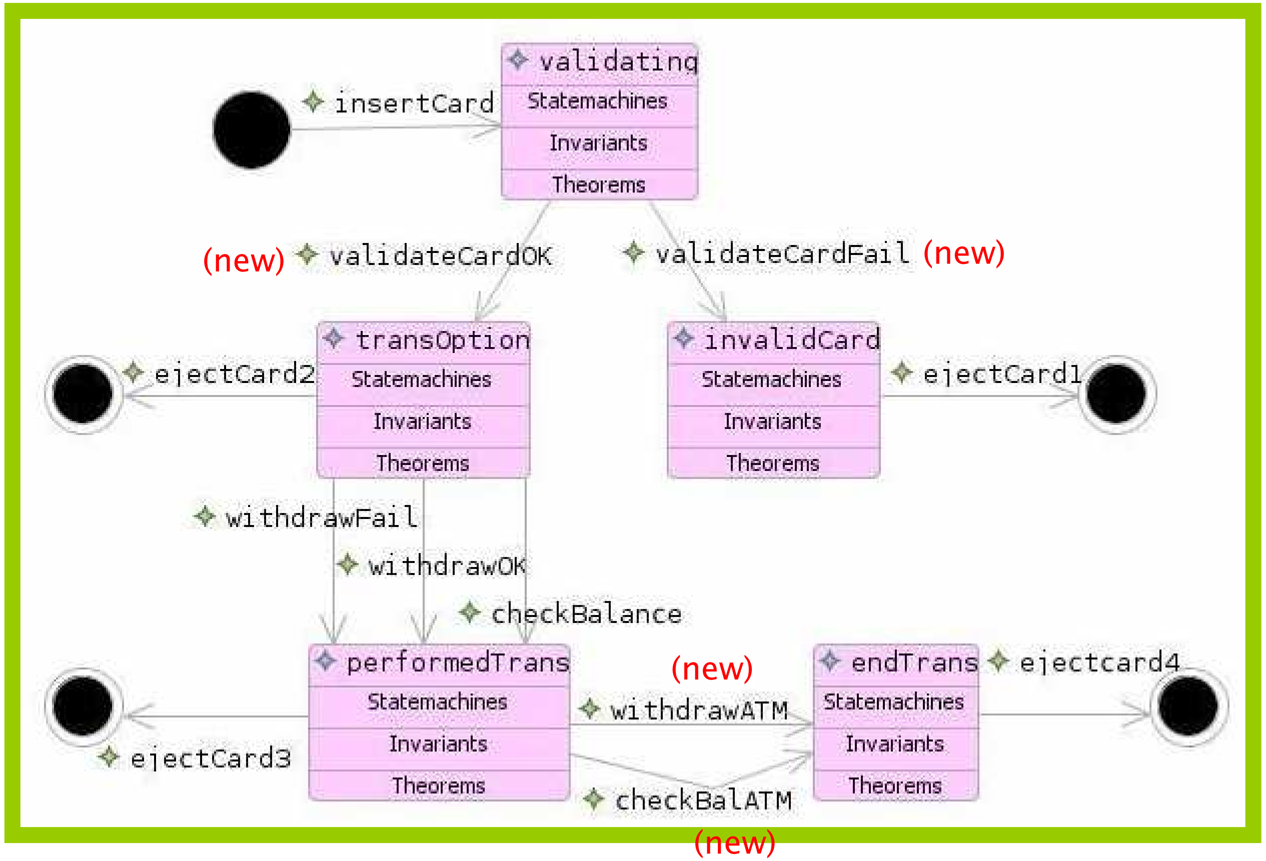
Four *ejectCard* events refining one abstract event



nested state machine

ATM Second Refinement:

Nested state machine of the refined state *active_atm*



Conclusion

- Introduced the notion of refined classes and refined state machines.
- Introduced five refinement techniques:
  - Add new classes in a refinement
  - Add new attributes and associations to a refined class
  - State elaboration
  - Transition elaboration
  - Move event in a refined class or a new class in a refinement
- The above techniques has been experimented in the ATM case study using the UML-B tool.
- The Rodin tool provers were used to generate and prove the proof obligations of the case study.

Conclusion and Future Work

- The approach of elaborating states with sub-states in refinement:
  - Supports and incremental refinement approach.
  - Supports modular reasoning by localising the invariants required for refinement proofs.
- Extend UML-B to support decomposition.
- Add support for parallel state machine.