

Composition Semantics for Executable and Evolvable Behavioral Modeling in MDA

BM MDA 2009

Ashley McNeile

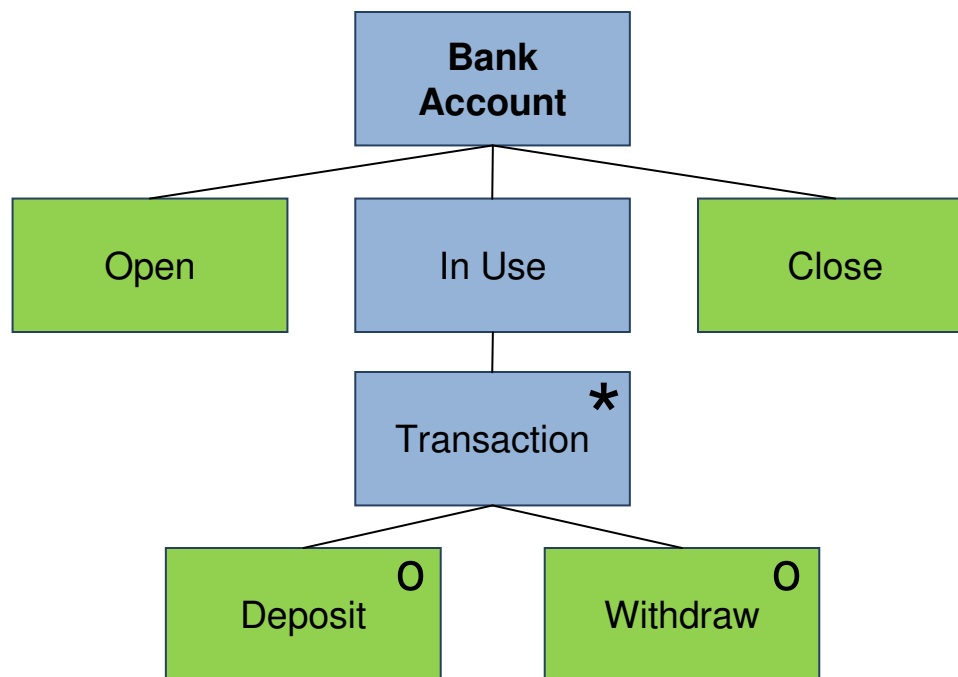
Metamaxim Ltd, UK

Ella Roubtsova

Open University of the Netherlands

Early ideas in OO (1980s) Domain Object Behaviour

- JSD (*Jackson System Development*)
- Shlaer/Mellor (*Recursive Design*)



M.A. Jackson

Diagrams used to describe and formalise “object life-cycles.”

Limited Expressive Power

Eclipsed by “OOP mania” in 1980s and 1990s.

Process Algebra

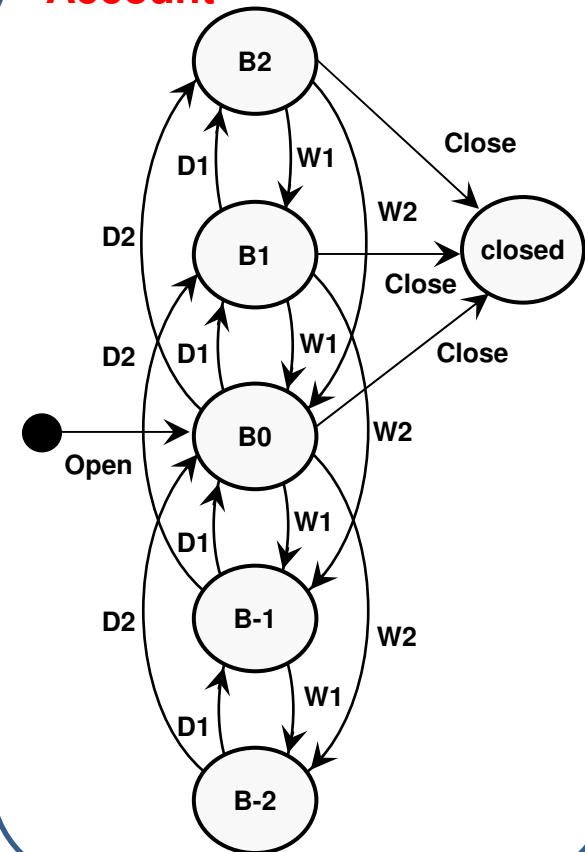
Account

Balance in range -2 to +2
Can Deposit or Withdraw 1 or 2 units at a time
Cannot Close if overdrawn

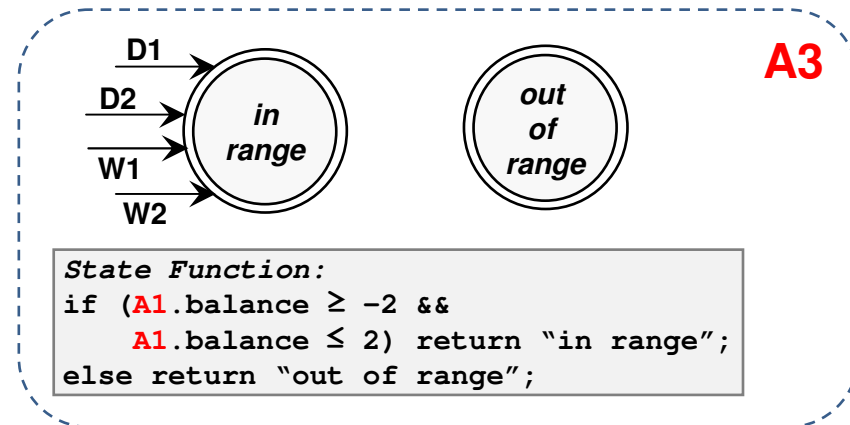
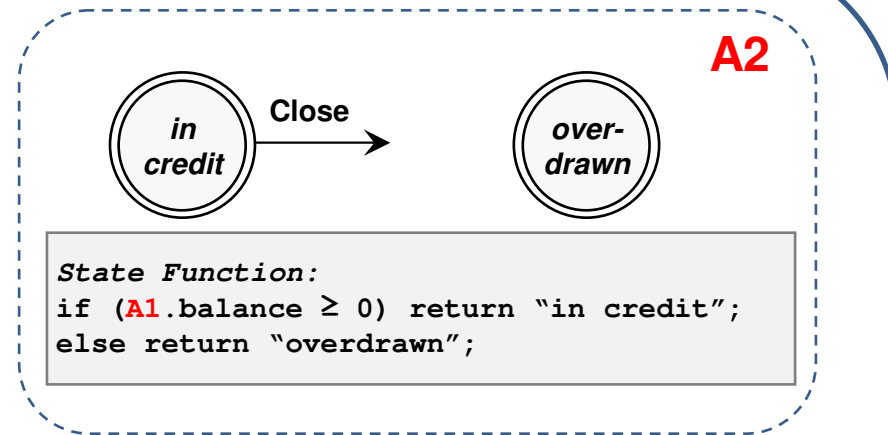
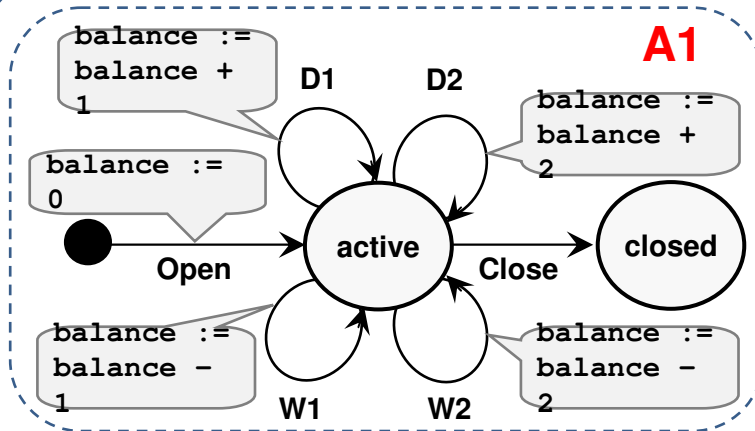
Account

$ACCOUNT = Open.B0$
 $B0 = D1.B1 + D2.B2 + W1.B-1 + W2.B-2 + Close.closed$
 $B1 = D1.B2 + W1.B0 + W2.B-1 + Close.closed$
 $B2 = W1.B1 + W2.B0 + Close.closed$
 $B-1 = D1.B0 + D2.B1 + W1.B-2$
 $B-2 = D1.B-1 + D2.B0$

Account



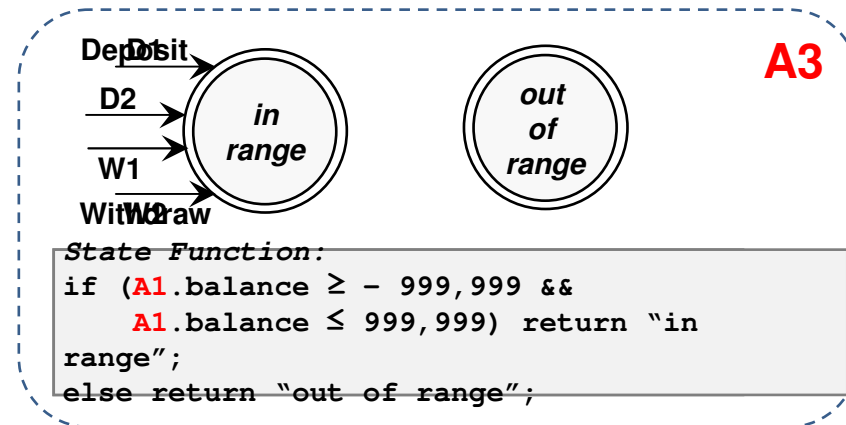
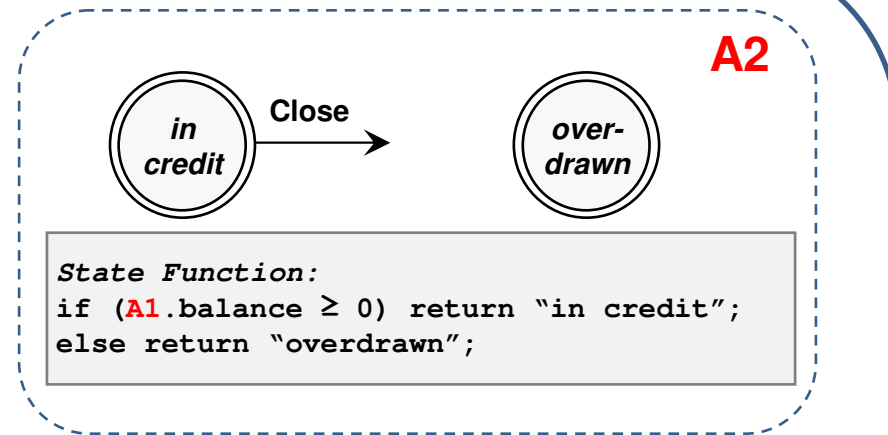
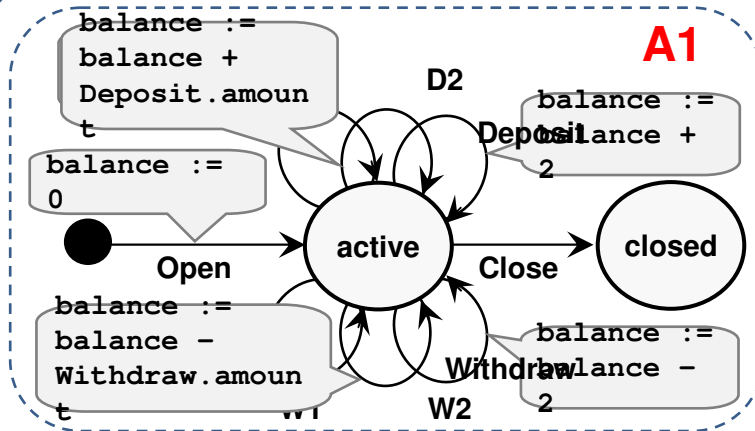
Protocol Modelling



ACCOUNT = A1 || A2 || A3

Identical "Black Box Behaviour" to the algebraic description

“Arbitrary” Balance



ACCOUNT = A1 || A2 || A3

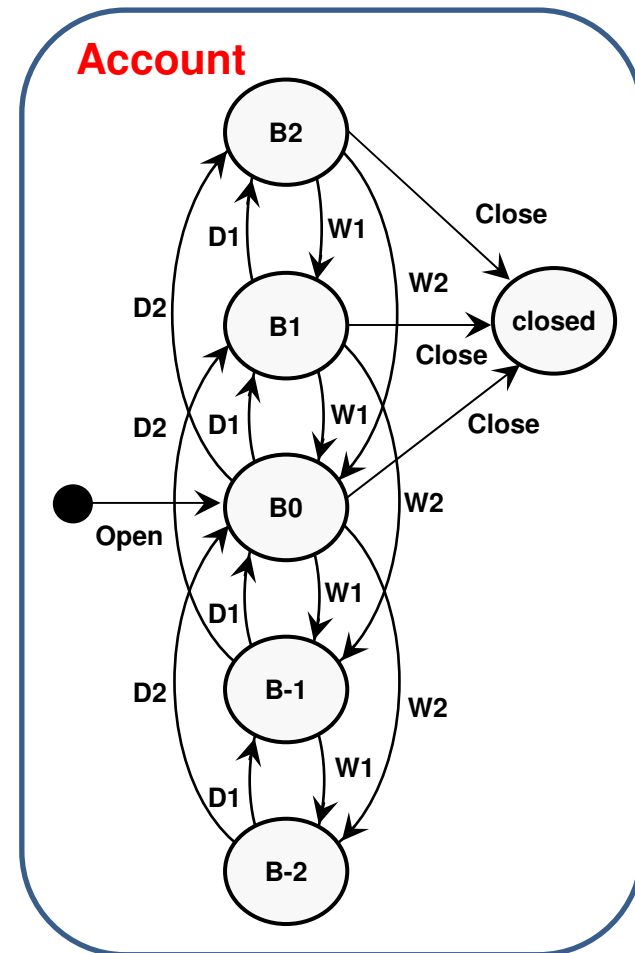
The Central Idea

How would you make this change to the algebraic description?

The combination of:

- Machines with local storage (balance)
 - Events with attributes (amount)
 - Derived states (in Credit, Overdrawn)
- gives the ability to model “real” systems.

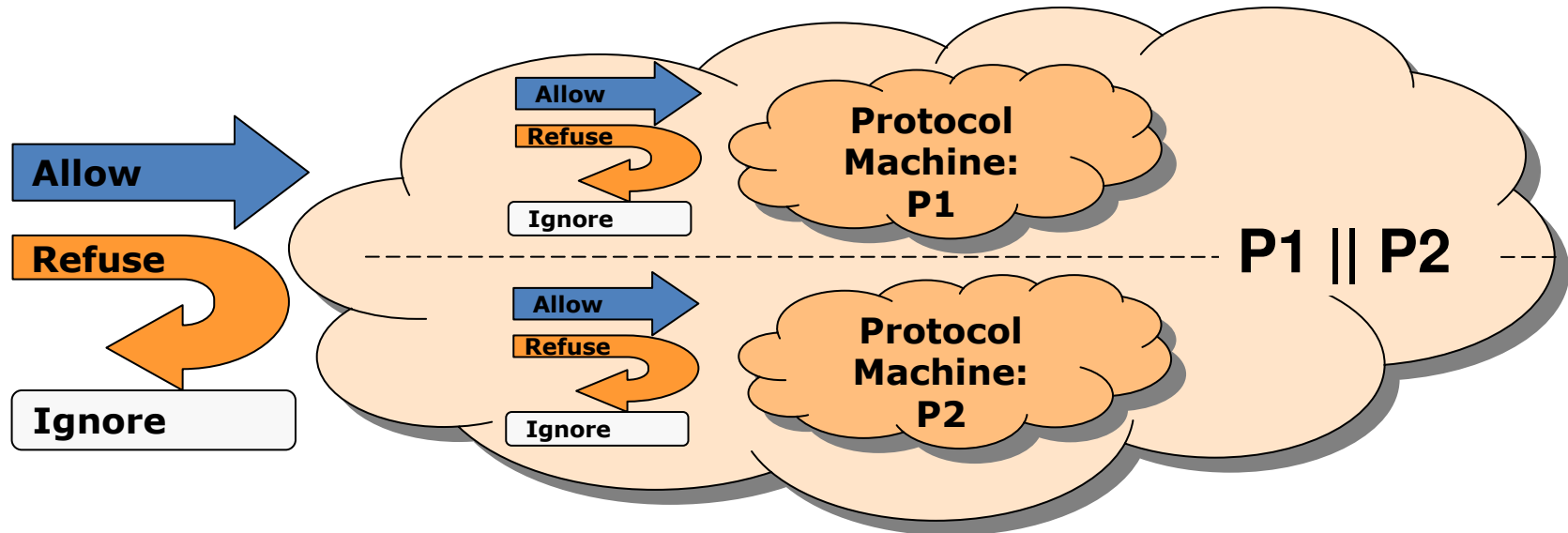
But we retain the ability to do process algebraic composition.



Protocol Machines

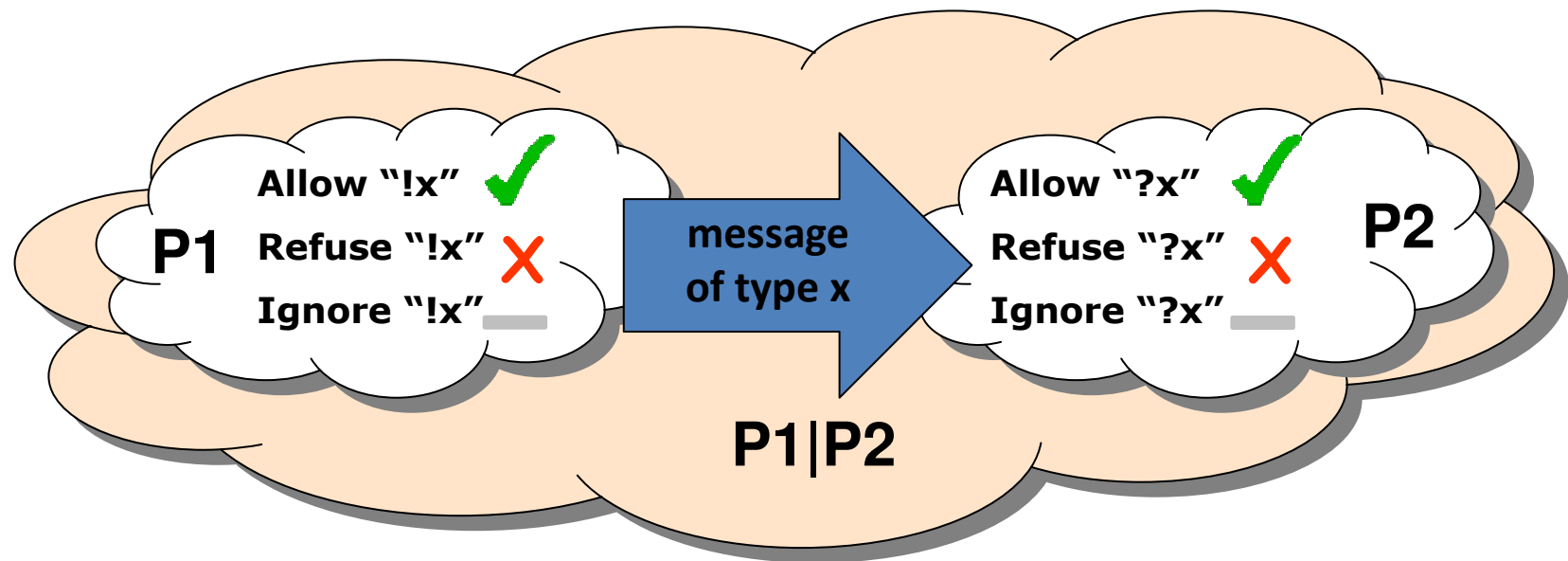
- Behaviour is modelled by *composing Protocol Machines*
- Machines are presented with *Events*
- Each machine has an *Alphabet* of events that it understands
- A machine will *Ignore* an event not in its alphabet, otherwise will *Allow* or *Refuse* based on its *State*

Composition (CSP)



Ignore	and	Ignore	=	Ignore
Refuse	or	Refuse	=	Refuse
Anything else			=	Allow

Composition (CCS)



P (!x)	Q (?x)	P Q
✓ and ✓		⇒ Message of type x can be exchanged
Anything else		⇒ Message of type x cannot be exchanged

Model Execution

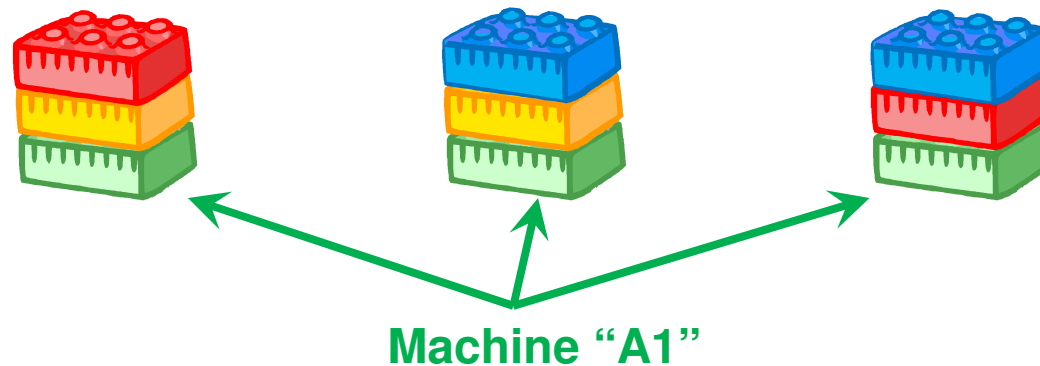
The screenshot shows a software interface for model execution, divided into several sections:

- Objects:** A list containing 'Account' and 'Customer'. A callout points to this list: "List of object types in the model."
- Attributes:** A table for 'Account' with values: 'Account Number 0002', 'Owner Ashley McNeile', and 'Balance -40.00'. A callout points to this section: "Attribute values of the selected instance."
- Events:** A list containing 'Cash Deposit', 'Cash Withdraw', and 'Freeze'. A callout points to this list: "Event types allowed by the selected instance."
- Instance:** A list containing '(new Account) 0001' and '0002'. A callout points to this list: "List of Instances of the selected object type."
- Form:** An input field for 'Amount' with the value '30.00' and a 'Cash Withdraw' button. A callout points to the button: "Entry of attribute values to submit an event – in this case, a 'Cash Withdraw'."
- Post-state Error Checking:** A section containing an error message: "Event 'Cash Withdraw' not possible: bad post-state 'over limit' in 'Account' 0002". A callout points to this section: "Button to submit the event to the model."

A large callout at the bottom of the interface reads: "Error message: 'Cash Withdraw' event has violated the post-state constraint".

“Composition vs Inheritance”

- Should we be creating *behavioural variants* by *Inheritance* or *Composition*?
- This work naturally points to *Composition*



THANK YOU

Presentation at BM MDA 2009