

# **DIMA-Maude: Toward a Formal Framework for Specifying and Validating DIMA Agents**

Farid Mokhati, Noura Boudiaf  
Mourad Badri & Linda Badri

# Outline

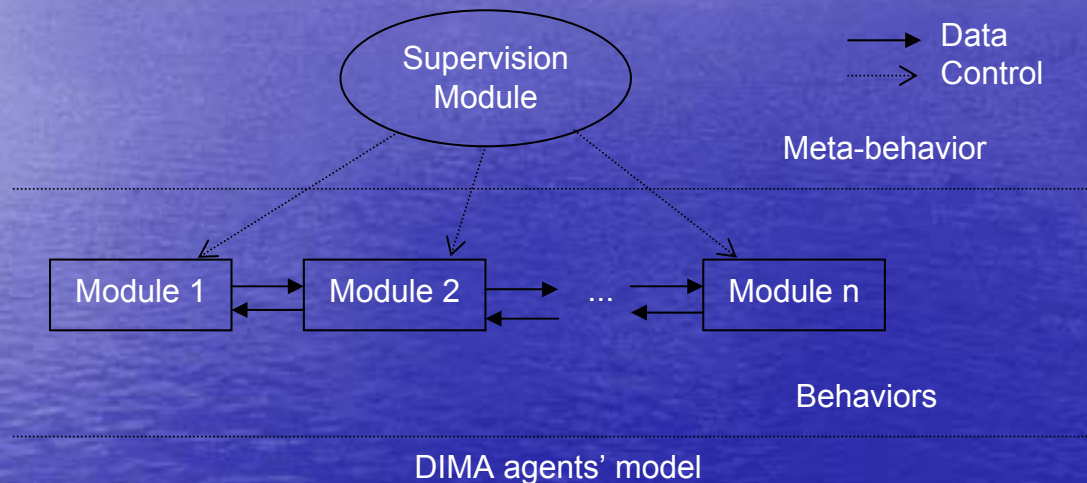
- Introduction
- DIMA multi-Agents Model
- Rewriting Logic & Maude
- Translation Process
- Example : Auction
- Validation Process
- Conclusion & Future Work

# Introduction

- MAS Behavior is mostly specified using diagrammatic or semi-formal methodologies.
- Approaches supporting a partial formalization of multi-agents systems.
- To reach consistency in their behavior, the MAS need to be clearly specified, validated and correctly implemented.
- Selected language : Maude.

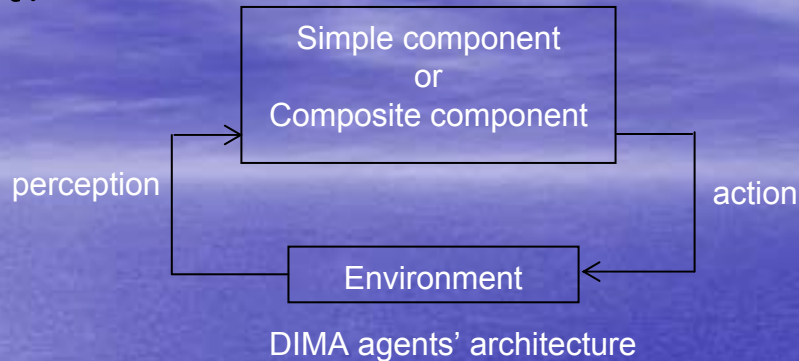
# DIMA multi-Agents' Model

- DIMA agents' model

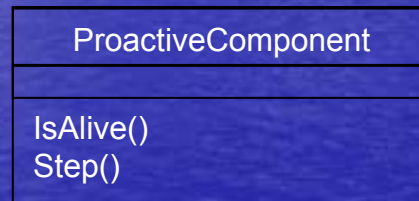


- DIMA proposes to decompose an agent in different components.
- Goal : integrating some existing paradigms.
- An agent can have one or several components that can be reactive or cognitive.

- An agent is a simple component, or a composite component.



- Basic brick = a proactive component



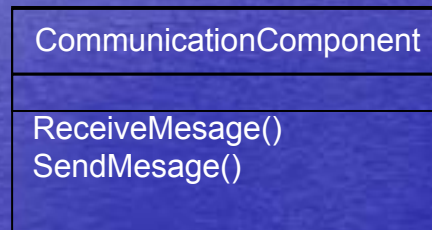
General structure of a proactive component

It describes :

- Goal: method *IsAlive ()*.
- Basic behaviors: as methods.
- Meta-behavior: method *Step ()*.

# ATN-Based Communicating Agents

- A communicating agent is a proactive agent that introduces new functionalities of communication.



General Structure of a communication component

- Meta-behavior is described by an ATN.
  - *IsAlive()* tests if the final state is not reached.
  - *Step()* allows to activate from the current state of an agent a transition whose condition is verified.

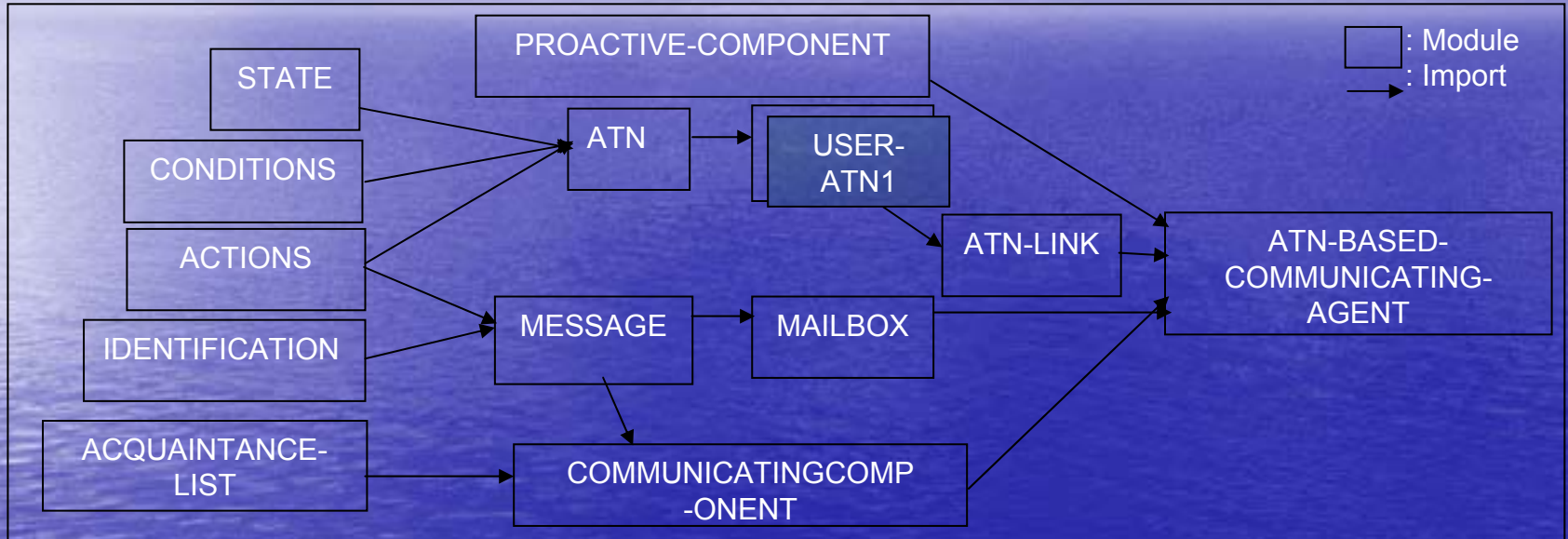
# Rewriting Logic

- Describes a concurrent system (states and transitions). Such a system is depicted by a rewriting theory  $T = (\Sigma, E, L, R)$ .
- Statique structure of system is described by the signature  $(\Sigma, E)$  that presents states of a system.
- Dynamique structure of system is described by rewriting rules  $rl [l] : t \rightarrow t'$ . These rules represent transitions.

# Maude

- Based on rewriting logic : expresses the concurrency and the change of states of the system.
- Large spectre : specification, prototypage, etc.
- Multi-paradigmes :it combines the functional and oriented-object programming.
- Maude Syntax : regroupe three types of modules :
  - Functional modules.
  - System modules.
  - Oriented-object modules.

# Translation process



DIMA-Maude frameworks' architecture.

```

(fmod STATE is
sorts State KindState NameState .                               ***[1]

ops initial final ordinary : -> KindState .                   ***[2]
op AgentState : NameState KindState -> State .                 ***[3]
op IsInitial : State -> Bool .                                  ***[4]
op IsOrdinary : State -> Bool .                                 ***[5]
op IsFinal : State -> Bool .                                   ***[6]

var k : KindState .
var ns : NameState .

eq IsInitial(AgentState(ns, k)) = if k == initial then true   ***[7]
                                else false fi .
eq IsOrdinary(AgentState(ns, k)) = if k == ordinary then true ***[8]
                                else false fi .
eq IsFinal(AgentState(ns, k)) = if k == final then true      ***[9]
                                else false fi .

endfm)

```

The functional module *STATE*.

```

(fmod CONDITIONS is
sorts Condition .
op NoCondition : -> Condition .
endfm)

```

The functional module *CONDITIONS*.

```

(fmod ACTIONS is
sort Action .
op NoAction : -> Action .
op IsInternalAction : Action -> Bool .
op IsSendingAction : Action -> Bool .
op IsReceivingAction : Action -> Bool .
op IsSendingActionToAll : Action -> Bool .
endfm)

```

The functional module *ACTIONS*.

```
(fmod ATN is
protecting STATE .
protecting CONDITIONS .
protecting ACTIONS
op TargetState : State Condition -> State .
op AccomplishedAction : State Condition -> Action .
endfm)
```

The functional module *ATN*.

```
(fmod USER-ATN is
extending ATN .
***User part***
endfm)
```

The functional module *USER-ATN*.

```
(fmod ATN-LINK is
protecting USER-ATN .
op CorrspondingAgentState : State -> State .
op CorrspondingCondition : Action ->Condition .
***User part***
endfm)
```

The functional module *ATN-LINK*.

```
(fmod IDENTIFICATION is
  sorts AgentIdentifier .
  subsort AgentIdentifier < Oid .
endfm)
```

The functional module *IDENTIFICATION*.

```
(fmod MESSAGE is
  protecting IDENTIFICATION .
  protecting ACTIONS .
  sorts Message Content .
  subsort Action < Content .
  op _:_ : AgentIdentifier Content
  AgentIdentifier -> Message .
endfm)
```

The functional module *MESSAGE*.

```
(fmod COMMUNICATION-COMPONENT is
  protecting ACQUAINTANCE-LIST .
  protecting MESSAGE .
  subsort acquaintance < AgentIdentifier .
  op SendMessage : Message -> Msg .
  op ReceiveMessage : Message -> Msg .
endfm)
```

The functional module *COMMUNICATION-COMPONENT*.

```
(fmod PROACTIVE-COMPONENT is
  sort Parameters Void .
  op IsAlive : Parameters -> Bool .
  op Step : Parameters -> Void .
endfm)
```

The functional module *PROACTIVE-COMPONENT*.

(omod ATN-BASED-COMMUNICATING-AGENT is  
protecting PROACTIVE-COMPONENT .  
protecting ATN-LINK. protecting COMMUNICATION-COMPONENT . protecting MAILBOX .  
subsort State < Paramaters .

class Agent | CurrentState : State, MBox : MailBox, AccList : acquaintanceList .       \*\*\*[1]  
Msg Event : AgentIdentifier State Condition -> Msg .                               \*\*\*[2]  
Msg Execute : Action -> Msg .   \*\*\*[3]  
Msg GetEvent : AgentIdentifier State Condition -> Msg .                       \*\*\*[4]

eq IsAlive (AgentState(NS, KS)) = IsFinal(AgentState(NS, KS)) .               \*\*\*[5]

\*\*\*\*\*First case\*\*\*\*\*

cr1 [StepCase1] : Event(A, S, Cond) < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
=> Execute(AccomplishedAction(S, Cond)) GetEvent(A, S, Cond)  
    < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
    if (IsAlive(S) == false) and  
        (IsInternalAction(AccomplishedAction(S, Cond)) == true) .

rl [StepCase11] : GetEvent(S, Cond) Execute(Act)  
    < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
=> < A : Agent | CurrentState : TargetState(S, Cond), MBox : MB, AccList : ACL > .

\*\*\*\*\*Second case\*\*\*\*\*

cr1 [StepCase2] : Event(A, S, Cond) < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
                    < A1 : Agent | CurrentState : S1, MBox : MB1, AccList : ACL1 >  
=> SendMessage(A : AccomplishedAction(S, Cond) : A1) GetEvent(A, S, Cond)  
    < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
    < A1 : Agent | CurrentState : S1, MBox : MB1, AccList : ACL1 >  
    if (IsAlive(S) == false) and  
        (IsSendingAction(AccomplishedAction(S, Cond)) == true) .

rl [StepCase21] : GetEvent(A, S, Cond) SendMessage(A : Act : A1)  
    < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >  
=> Event(A1, CorrespondingAgentState(S), CorrspondingCondition(Act))  
    < A : Agent | CurrentState : TargetState(S, Cond), MBox : MB, AccList : ACL > .

```

*****Third case*****
crl [StepCase3] : Event(A, S, Cond) < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >
    => ReceiveMessage(A1 : AccomplishedAction(S, Cond) : A ) GetEvent(A, S, Cond)
        < A : Agent | CurrentState : S, MBox : InsertInMBox( (A1 : AccomplishedAction(S, Cond) : A), MB), AccList : ACL >
        if (IsAlive(S) == false) and (IsReceivingAction(AccomplishedAction(S, Cond)) == true) .

rl [StepCase31] : GetEvent(A, S, Cond) ReceiveMessage(A1 : Act : A)
    < A : Agent | CurrentState : S, MBox : MB, AccList : ACL >
    => < A : Agent | CurrentState : TargetState(S, Cond), MBox : MB, AccList : ACL > .

*****Fourth case*****
crl [IniSendingToAllAct] : Event(A, S, Cond) < A : Agent | CurrentAgentState : S, MBox : MB, AccList : ACL >
    =>
        SendMessage(A : AccomplishedAction(S, Cond) : HeadA(ACL)) GetEvent(A, S, Cond)
        < A : Agent | CurrentAgentState : S, MBox : MB, AccList : TailA(ACL) >
        if (IsAlive(S) == false) and (IsSendingActionToAll(AccomplishedAction(S, Cond)) == true)
        and (Head(ACL) /= Erroracquaintance) .

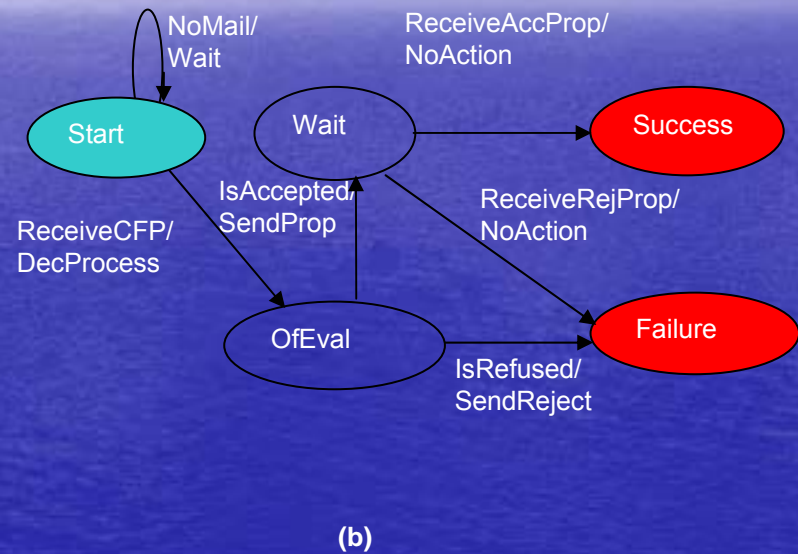
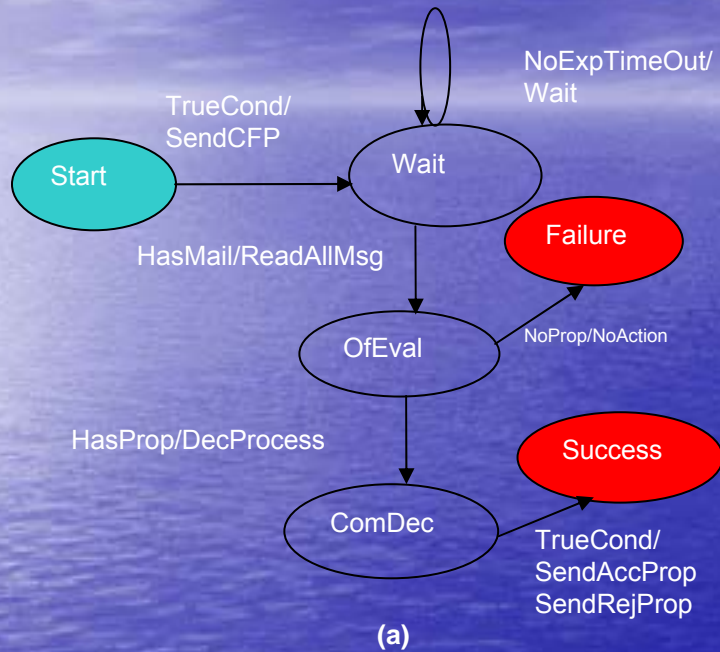
crl [IniSendingToAllAct] : GetEvent(A, S, Cond) SendMessage(A : Act : A1)
    < A : Agent | CurrentAgentState : S, MBox : MB, AccList : ACL >
    =>
        Event(A, S, Cond)
        < A : Agent | CurrentAgentState : S, MBox : MB, AccList : ACL >
        Event(A1, CorrespondingAgentState(S), CorrespondingCondition(Act))
        if Head(ACL) /= Erroracquaintance .

...
endom)

```

The object-oriented module *ATN-BASED-COMMUNICATING-AGENT*.

# Example : Auction



ATNs that describe respectively the meta-behavior of the Auctioneer and the Bidder.

```

(fmod USER-ATN1 is
extending ATN .

***definition of states constituting the ATN*****

ops StartI WaitI OfferEvaluationI CommitmentDecision SuccessI FailureI : -> NameState .      ***[1]
ops AgentState(StartI, initial) AgentState(WaitI, ordinary) AgentState(OfferEvaluationI, ordinary)
  AgentState(CommitmentDecisionI, ordinary) AgentState(SuccessI, final)
  AgentState(FailureI, final) : -> State .                                               ***[2]

***definition of conditions and actions of the ATN*****
ops TrueCondition NoExpiredTimeOutI HasMail HasProposal NoProposal : -> Condition .      ***[3]
ops SendCFP WaitI ReadAllMsg DecisionProcessI SendAccpetProposal
  SendRejectProposal : -> Action .                                                       ***[4]

*** determination of the target state according to a state source and a condition *****

eq TargetState(AgentState(StartI, initial), TrueCondition) = AgentState(WaitI, ordinary) .  ***[5]
...
*** determination of the action to execute according to a state and a condition *****

eq AccomplishedAction(AgentState(StartI, initial), TrueCondition) = SendCFP .           ***[6]
...
***Determination of the type of an action*****

eq IsInternalAction(DecisionProcessI) = true .                                         ***[7]
...
eq IsSendingActionToAll(SendCFP) = true .                                             ***[8]
...
endfm)

```

The module USE-ATN corresponding to the agent Auctioneer

```

(fmod USER-ATN2 is
extending ATN .

***definition of states constituting the ATN*****

ops StartP WaitP OfferEvaluationP SuccessP FailureP : -> NameState .
ops AgentState(StartP, initial) AgentState(WaitP, ordinary) AgentState(OfferEvaluationP, ordinary)
  AgentState(SuccessP, final) AgentState(FailureP, final) : -> State .

***definition of conditions and actions of the ATN*****

ops ReceiveCFP NoMail IsAccepted IsRefused ReceiveAcceptProposal ReceiveRejectProposal : -> Condition
ops DecisionProcessP WaitP SendPropose RejectPropose :-> Action .

*** determination of the target state according to a state source and a condition *****

eq TargetState(AgentState(StartP, initial), ReceiveCFP) = AgentState(OfferEvaluationP, ordinary) .
...
*** determination of the action to execute according to a state and a condition *****

eq AccomplishedAction(AgentState(StartP, initial), ReceiveCFP) = DecisionProcessP .
...
***Determination of the type of an action*****

eq IsInternalAction(DecisionProcessP) = true .
...
eq IsSendingAction(SendPropose) = true .
...
endfm)

```

The module USE-ATN corresponding to the agent Bidder.

```

fmod ATN-LINK is
protecting USER-ATN1 .
protecting USER-ATN2 .

op CorrespondingAgentState : State -> State .
op CorrespondingCondition : Action -> Condition .

***** Auctionner part *****

eq CorrespondingState(AgentState(startI, initial)) = AgentState(startP, initial) .          ***[1]
eq CorrespondingState(AgentState(commitmentdecisionI, ordinary)) = AgentState(waitP, ordinary) .
...
eq CorrespondingCondition(SendCFP) = ReceiveCFP .                                          ***[2]
eq CorrespondingCondition(SendAcceptProposal) = ReceiveAcceptProposal .
...

***** Bidder part *****

eq CorrespondingState(AgentState(waitP, ordinary)) = AgentState(offerEvaluationI, ordinary) .  ***[3]
eq CorrespondingState(AgentState(offerEvaluationP, ordinary)) = AgentState(waitI, ordinary) .
...
eq CorrespondingCondition(SendPropose) = HasProposal .                                    ***[4]
eq CorrespondingCondition(SendReject) = NoProposal .

...
endfm

```

The module ATN-LINK .

# Validation process

**First case** : We suppose that all three Bidders accept to submit the proposal.

```
< "Auctioneer" : Agent | CurrentAgentState : AgentState(startI, initial), MBox : EmptyMailBox, AccList : ("Bidder1" : ("Bidder2" : "Bidder3")) >
< "Bidder1" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >
< "Bidder2" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >
< "Bidder3" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >
Event("Auctioneer", AgentState(startI, initial), TrueCondition)
Event("Bidder1", AgentState(offerEvaluationP, ordinary), IsAccepted)
Event("Bidder2", AgentState(offerEvaluationP, ordinary), IsAccepted)
Event("Bidder3", AgentState(offerEvaluationP, ordinary), IsAccepted) .
```

The Auctioneer launches its evaluation process and all Bidders wait the result.

```
< "Auctioneer" : Agent | CurrentAgentState : AgentState(OfferEvaluationI, ordinary), MBox : EmptyMailBox, AccList : ("Bidder1" : ("Bidder2" : "Bidder3")) >
< "Bidder1" : Agent | CurrentAgentState : AgentState(waitP, ordinary), MBox : MB1, AccList : "Auctioneer" >
< "Bidder2" : Agent | CurrentAgentState : AgentState(waitP, ordinary), MBox : MB1, AccList : "Auctioneer" >
< "Bidder3" : Agent | CurrentAgentState : AgentState(waitP, ordinary), MBox : MB1, AccList : "Auctioneer" >
```

Second case : We suppose that two Bidders accept to submit proposals and one refuses to submit...

```
< "Auctioneer" : Agent | CurrentAgentState : AgentState(startI, initial), MBox : EmptyMailBox, AccList : ("Bidder1" :  
                                                                    ("Bidder2" : "Bidder3")) >  
< "Bidder1" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >  
< "Bidder2" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >  
< "Bidder3" : Agent | CurrentAgentState : AgentState(startP, initial), MBox : MB1, AccList : "Auctioneer" >  
Event("Auctioneer", AgentState(startI, initial), TrueCondition)  
Event("Bidder1", AgentState(offerEvaluationP, ordinary), IsAccepted)  
Event("Bidder2", AgentState(offerEvaluationP, ordinary), IsAccepted)  
Event("Bidder3", AgentState(offerEvaluationP, ordinary), IsRefused) .
```

The Auctioneer launches its evaluation process and two Bidders wait the result and the other finish in failure.

```
< "Auctioneer" : Agent | CurrentAgentState : AgentState(offerEvaluationI, ordinary), MBox : EmptyMailBox, AccList :  
                                                                    ("Bidder1" : ("Bidder2" : "Bidder3")) >  
< "Bidder1" : Agent | CurrentAgentState : AgentState(waitP, ordinary), MBox : MB1, AccList : "Auctioneer" >  
< "Bidder2" : Agent | CurrentAgentState : AgentState(waitP, ordinary), MBox : MB1, AccList : "Auctioneer" >  
< "Bidder3" : Agent | CurrentAgentState : AgentState(FailureP, final), MBox : MB1, AccList : "Auctioneer" >
```

# Conclusions

- We presented a formal framework called DIMA-Maude for supporting formal description and validation of DIMA model's agents.
- The framework is described using the formal and object-oriented language Maude.
- The developed framework, in its present version, is extensible.
- It is composed of several modules representing the basic bricks for formal description and validation of DIMA communicating agents whose meta-behavior is described by an ATN.

## perspectives

- Verifying some properties of the specification of the DIMA model described in Maude using Model Checker.
- Reuse some part of this framework to support Rule-based framework and Case-based framework.

The background is a smooth blue gradient. On the left side, there is a bright, glowing area that resembles a sun or moon reflecting on a body of water, with a soft, white-to-yellow glow that fades into the blue. The overall effect is serene and clean.

Thank You