Specification and Verification of a Dynamic Reconfiguration Protocol for Agent-Based Applications

Manuel Aguilar Cornejo, Hubert Garavel, Radu Mateescu, Noël de Palma

INRIA Rhône-Alpes / VASY

Outline

- Introduction
- Dynamic reconfiguration protocol
- Formal specification in LOTOS
- Verification using the CADP toolbox
- Conclusion and future work

Introduction

- **Context of the work:**
  - message-oriented middlewares (*MOM*)
  - agent-based applications
- **Cooperation between INRIA and Bull:**
  - AAA (*Agents Anytime Anywhere*) middleware
    - dynamic reconfiguration features
  - Netwall security product of Bull
    - multiple firewall coordination, log auditing
- **Objective:**

  validate AAA's dynamic reconfiguration protocol

AAA distributed agent model

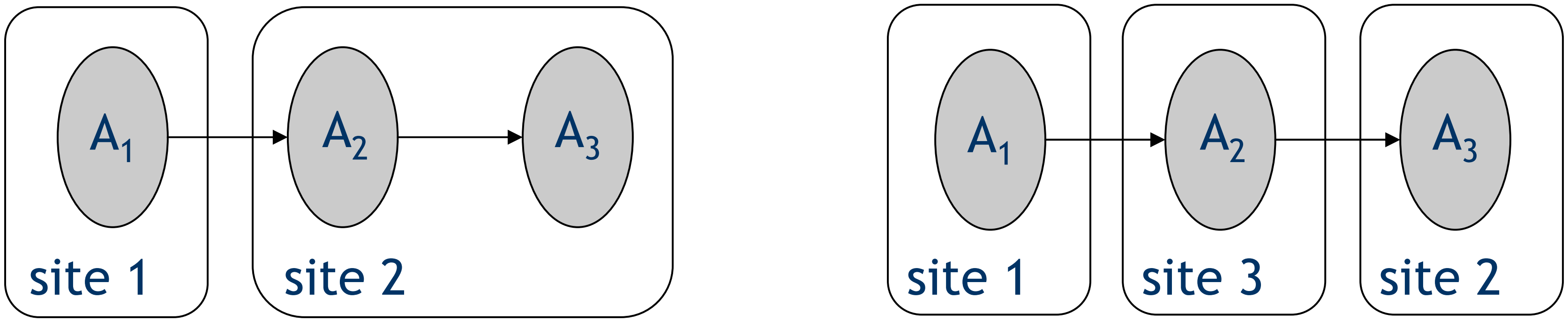
- **Agents:**
  - sequential entities communicating by messages
  - execution using an event-reaction model
  - persistency, atomicity, configurability
- **Communication:**
  - unidirectional point-to-point channels
  - asynchrony, reliability, causality
- **Application:**
  - set of agents executing on several sites
  - communication channels between agents

Dynamic reconfiguration

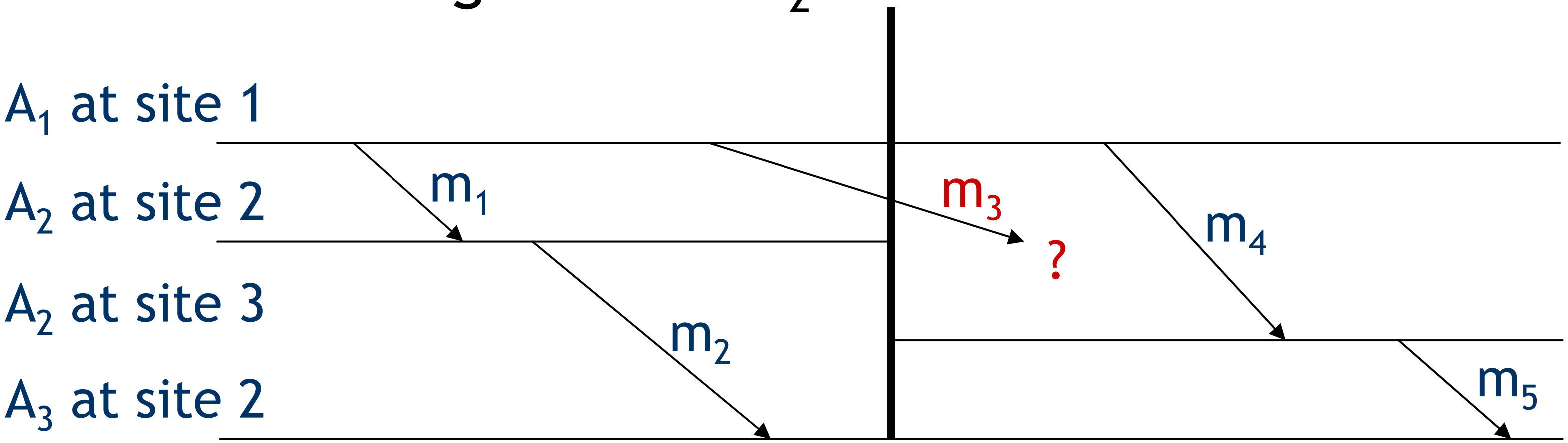
- **Run-time modifications of the application:**
  - Architecture (creation/deletion of agents, modification of communication channels)
  - Migration (placement of agents on sites)
  - Implementation (replacement of code)
  - Interface (upgrade of services)
- **Problem:**

  preserve the consistency of the application after reconfiguration

Inconsistency after migration



Migration of A_2 from site 2 to site 3



Avoiding inconsistencies

Three issues:

- Agent naming

  references to migrating agents must be properly updated

- Agent states

  agents must resume computation from their state prior to reconfiguration

- Communication channels

  messages in transit during reconfiguration must be preserved and properly redirected

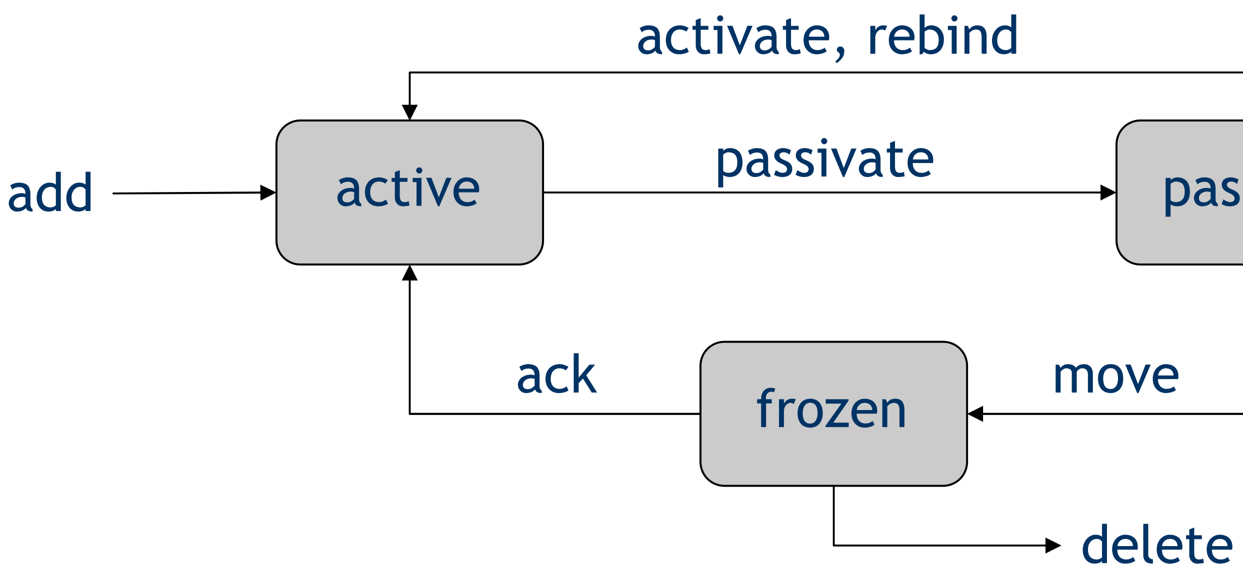
Principles of the protocol

- **Use a configurator agent, which:**
  - keeps a view of the application configuration
    - placement of agents on execution sites
    - communication channels between agents
  - handles all reconfiguration commands
    - ADD, DELETE, MOVE, BIND, REBIND
  - updates the configuration view accordingly
- **Precondition for safe reconfiguration:**

  all communication channels involved must be empty before the reconfiguration can occur

Abstract states of agents

- **Application agents can be:**
  - Active (execute normally)
  - Passive (react to events, but send no messages)
  - Frozen (cannot receive any event)



Overview of the protocol

Reconfiguration of an agent *A*:

1. Compute the *change passive set* of *A*
   cps(*A*) = agents with channels towards *A*
2. Passivate all agents in *cps*(*A*);
   when this is completed, *A* is frozen
3. Send the reconfiguration command to *A*;
   all channels towards *A* are empty
4. Activate all agents in *cps*(*A*)

Formal specification

- **LOTOS [ISO 1988]:**

  formal description technique for communication protocols and distributed systems

- **Two « orthogonal » sub-languages:**

  Data part (abstract data types, ActOne)

  - sorts and operations
  - equations and pattern-matching

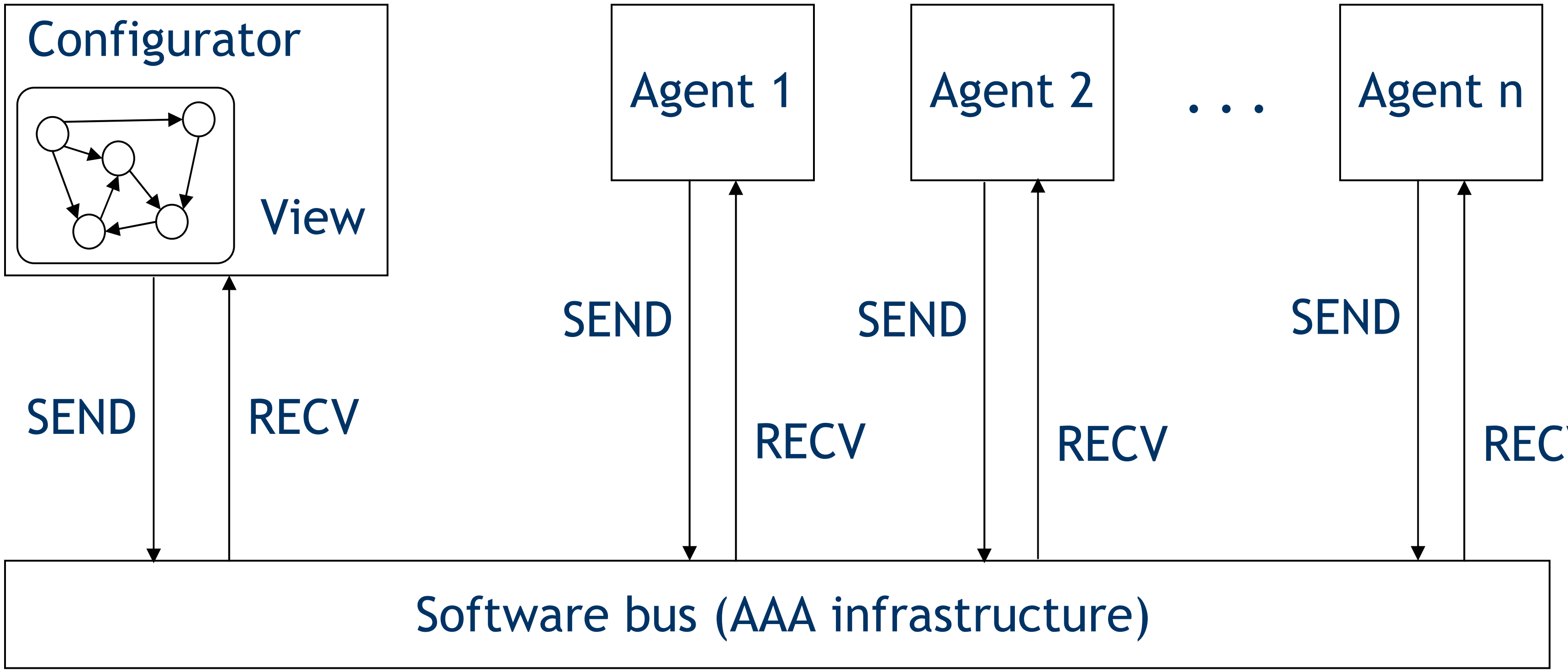
  Behaviour part (process algebras, CCS and CSP)

  - parallel processes interacting by rendez-vous
  - value-passing communication on gates

LOTOS - behaviour part

Behaviour operator	Meaning
stop	inaction
$G\ !V\ ?X{:}S\ ;\ B$	action prefix
B_1 [] B_2	choice
[E] -> B	conditional
B_1 |[$G_1,\ldots,G_n$]| B_2	parallel composition
B_1 ||| B_2	interleaving
exit	successful termination
B_1 >> B_2	sequential composition
P [$G_1,\ldots,G_n$] ($V_1,\ldots,V_m$)	process call

Architecture of the protocol



Architecture (in LOTOS)

```
behaviour
(
   Agent [SEND, RECV] (DEAD, a1@s1, {})
   |||
   . . .
   |||
   Agent [SEND, RECV] (DEAD, an@sn, {})
   |||
   Configurator [SEND, RECV] (nil, a1@s1+ … + an@sn + {})
)
|[SEND, RECV]|
Bus [SEND, RECV] (nil)
```

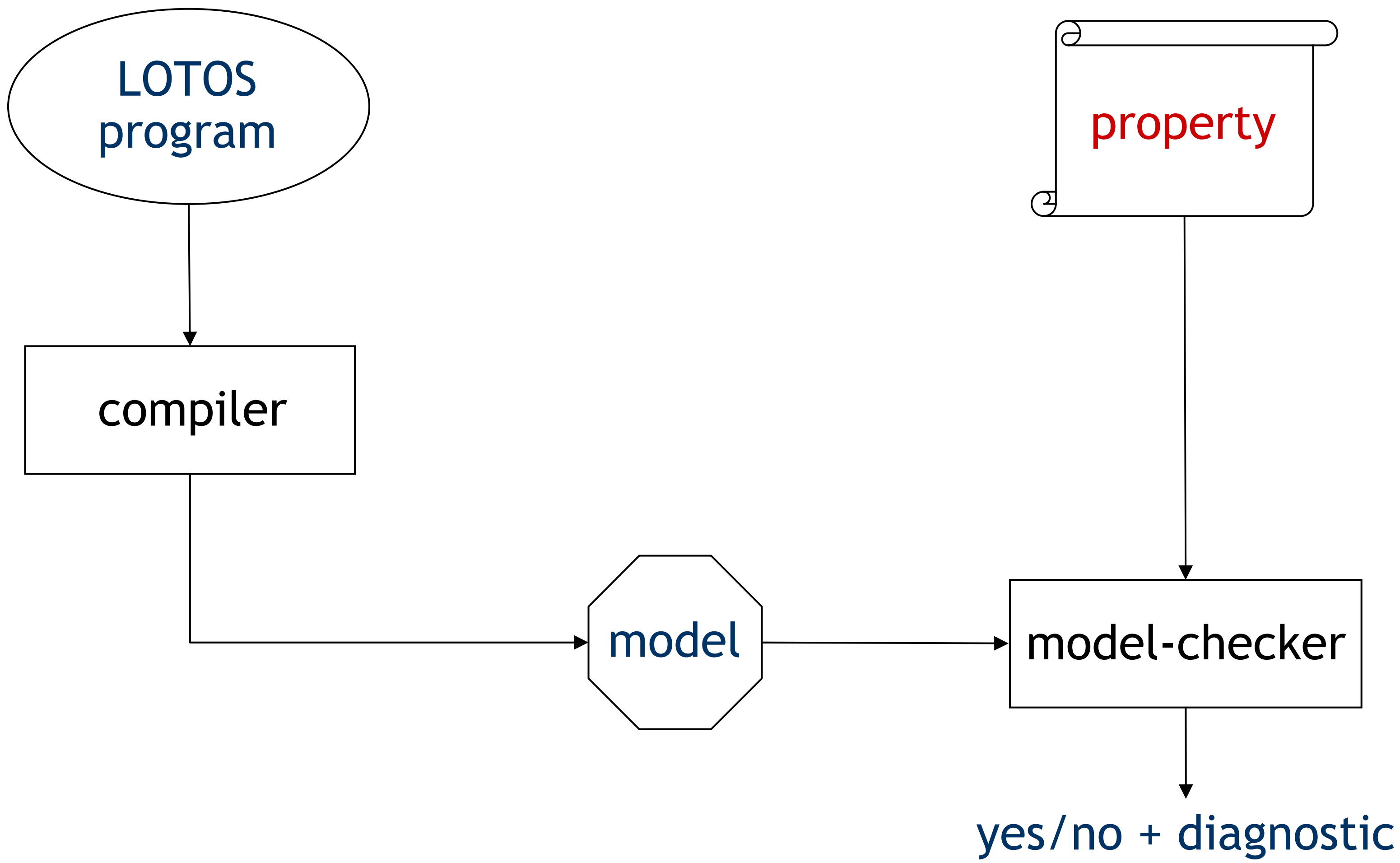
Configurator agent (in LOTOS)

```
process Configurator [SEND,RECV] (C:Config, R:AddrSet):=
  (* ADD command: add an agent to the application *)
  choice A:Addr []
    [(A notin C) and (A isin R)] ->
      SEND !A !confaddr !ADD !dummy !dummy;
        RECV !confaddr !A !ACK !dummy !dummy;
          Configurator [SEND,RECV] (ins(A & {}, C), rem(A,R))
  []
  (* . . . other reconfiguration commands *)
endproc
```

Application agent (in LOTOS)

```
process Agent [SEND,RECV] (S:State, A:Addr, R:Addrset):=
  [S eq DEAD] ->
    RECV !A !confaddr !ADD !dummy !dummy;
      SEND !confaddr !A !ACK !dummy ! dummy;
        Agent [SEND, RECV] (ACTIVE, A, {})
  []
  (* … other reconfiguration commands *)
endproc
```

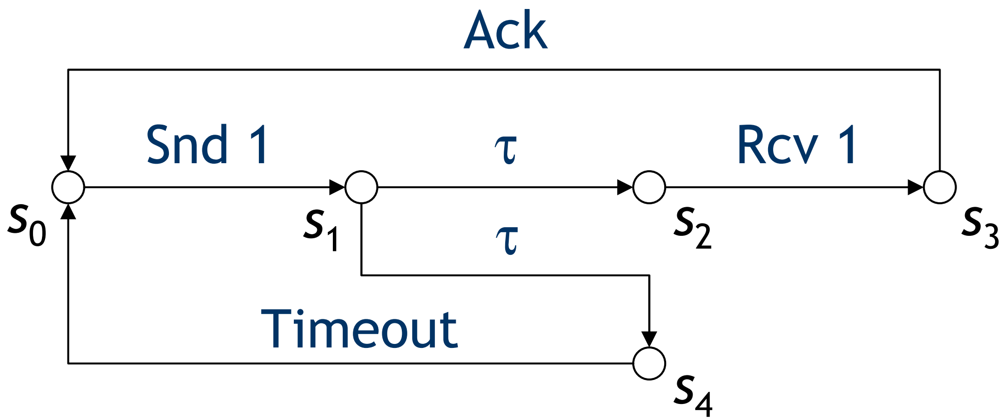
Verification by model-checking



Model

Labelled Transition System (S, A, T, s_0):

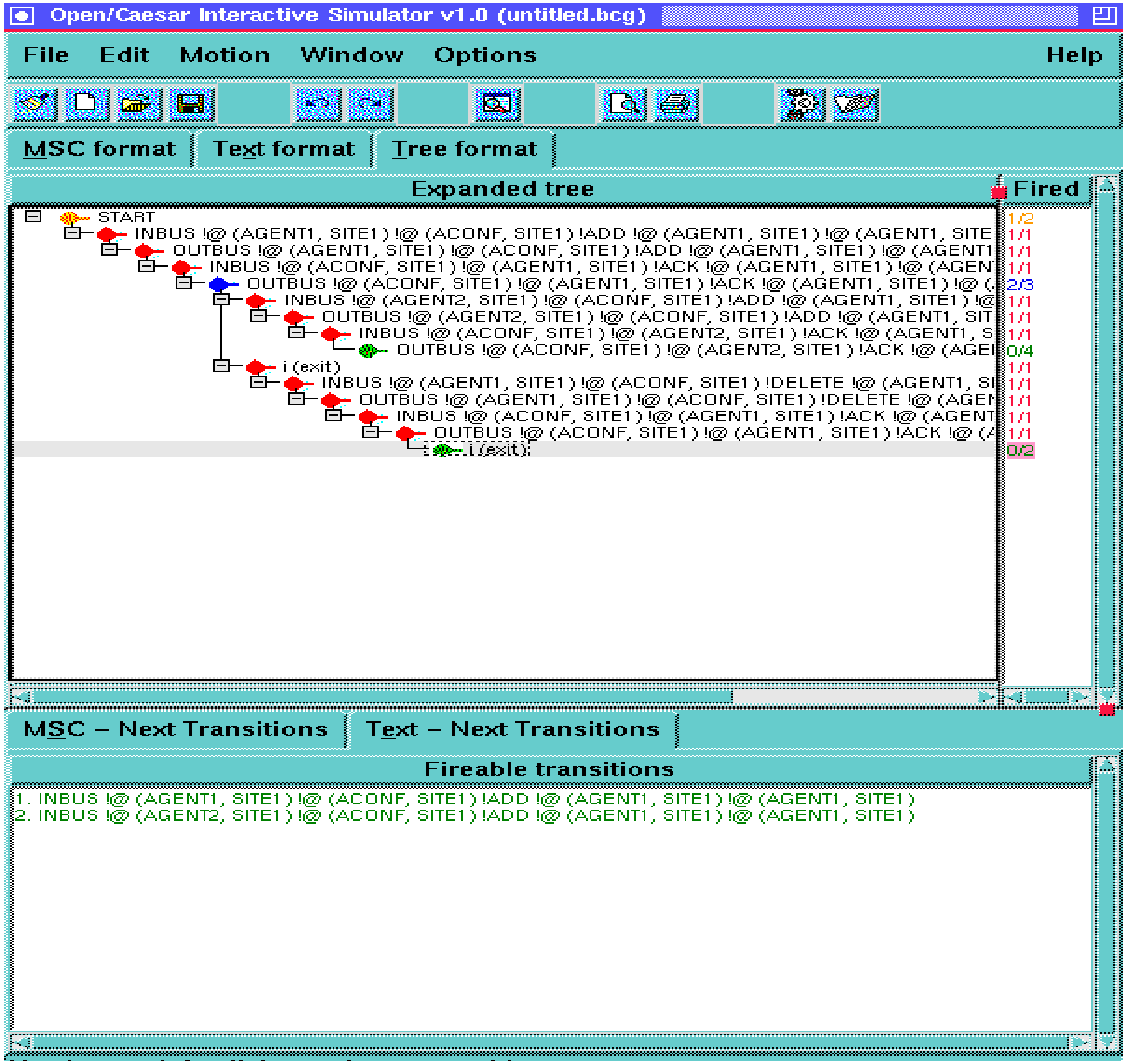
- S is the set of *states*
- A is the set of *actions* ($a = G\ v_1 \ldots v_n$)
- $T \subseteq S \times A \times S$ is the *transition relation*
- $s_0 \in S$ is the *initial state*



CADP
(http://www.inrialpes.fr/vasy/cadp)

- **Caesar/Aldebaran Development Package:**
  a toolbox for the verification of communication protocols and distributed systems
- **Functionalities:**
  - compilation (Caesar.adt, Caesar)
  - interactive and guided simulation (Ocis)
  - bisimulation checking (Aldebaran, Bcg_min)
  - temporal logic model-checking (Evaluator)
  - compositional verification (Svl)
  - test generation (Tgv)

Interactive simulation



Temporal logic

Regular alternation-free μ-calculus:

- *Action formulas* (ACTL):

$$\alpha ::= a \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2$$

- *Regular formulas* (PDL):

$$\beta ::= \alpha \mid \beta_1 \,.\, \beta_2 \mid \beta_1 \,\text{|}\, \beta_2 \mid \beta^*$$

- *State formulas* (μ-calculus):

$$\varphi ::= \mathrm{F} \mid \mathrm{T} \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \langle \beta \rangle \varphi \mid [\, \beta \,]\, \varphi \mid Y \mid \mu Y \,.\, \varphi \mid \nu Y \,.\, \varphi$$

Correctness properties

- **Safety:** something bad never happens

  After a move command, the target agent cannot receive any event until it completes its migration

  [«Rec !A !Move». (¬«Rec !A !Ack»)*. «Rec !A !any»] false

- **Liveness:** something good eventually happens

  Every reconfiguration command is eventually followed by an acknowledgement

  [«Snd !A !Cmd»] μX . (<true> true $\wedge$ [¬«Rec !A !Ack»] X)

Verification results

- Several experiments
  - bounded number of agents
  - bounded number of sites
  - various subsets of reconfiguration commands
- Successful check of 10 temporal properties
- Rapid growth of model size
  - exponential number of possible configurations

    3 agents, Add, Bind, Rebind, Move

    $\Rightarrow$ more than 1,000,000 states

Conclusion and future work

- **Formal specification and verification of AAA's middleware dynamic reconfiguration protocol:**
  - 900 lines of LOTOS specification
  - 10 safety and liveness properties
  - verification of several finite-state configurations
- **Future work:**
  - implement a distributed configurator agent
  - continue the validation on larger configurations
  - improve the tools (massively parallel model-checking)
  - automatic test generation