

Model Checking Applied to Mobile Agent System

Cyril Dumont and Fabrice Mourlin

LACL, Team Systems Specification and Verification

SAFA Workshop - October 6, 2010

Outline

- 1 Introduction
 - Mobile Agent System
 - Formal Specification
- 2 From π -calculus terms to timed automata
- 3 A case study : MCA Architecture
 - Presentation
 - Modeling
 - Verification
- 4 Conclusion and future works

Outline

- 1 Introduction
 - Mobile Agent System
 - Formal Specification
- 2 From π -calculus terms to timed automata
- 3 A case study : MCA Architecture
- 4 Conclusion and future works

Mobile Agent

Definition

A mobile agent is a program that can migrate from machine to machine in a heterogeneous network. This migration is a part of its execution.

Mobile Agent

Definition

A mobile agent is a program that can migrate from machine to machine in a heterogeneous network. This migration is a part of its execution.

- Some advantages over the traditional client/server model
 - Efficiency
 - Fault tolerance
 - Customization

High Order π -calculus

To model a mobile agent system, we use a [process algebra](#) as the language [\$\pi\$ -calculus](#) which this is the syntax :

Definition $A, \dots ::= A(x_1, \dots, x_n) = P$

Process $P, Q, \dots ::= 0 \mid \alpha. P \mid P \mid Q \mid P + Q \mid [a = b] P$
 $\mid [a = b] P \mid (\nu n) P \mid A(v_1, \dots, v_n)$

Prefix $\alpha, \dots ::= \tau \mid \bar{c} \langle a \rangle \mid c(x)$

Expression of mobile agent needs use of languages which allow higher order term, that's why we use the [High Order \$\pi\$ -calculus](#).

Prefix $\alpha, \dots ::= \tau \mid \bar{c} \langle Q \rangle \mid c(P)$

Definition of a mobile agent

- A **mobile agent** is defined as a standard agent

$$M \stackrel{def}{=} \tau . M' . 0$$

Definition of a mobile agent

- A **mobile agent** is defined as a standard agent

$$M \stackrel{\text{def}}{=} \tau . M' . 0$$

- Its **execution context** defines his mobile feature

$$\text{System} \stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \begin{cases} Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

Definition of a mobile agent

- A **mobile agent** is defined as a standard agent

$$M \stackrel{\text{def}}{=} \tau . M' . 0$$

- Its **execution context** defines his mobile feature

$$\text{System} \stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \begin{cases} Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

- Q sends M to P along the x channel : M is a mobile agent.

$$(\nu x) \left(\left(x(A) . A . P' \right) \mid \left(\bar{x} \langle M \rangle . Q' \right) \right)$$

π -calculus, sources of infinity

- Despite its simple syntax, the language has a **rich semantic**.
- Modeling in this process algebra leads to the definition of **infinite systems**
- Numerous sources of infinity :
 - **Recursion and parallelism** : Dynamic creation of processes, Dynamic evolution of systems
 - **Parameterized systems** : System sizing, Open Systems

π -calculus, sources of infinity

- Despite its simple syntax, the language has a **rich semantic**.
- Modeling in this process algebra leads to the definition of **infinite systems**
- Numerous sources of infinity :
 - **Recursion and parallelism** : Dynamic creation of processes, Dynamic evolution of systems
 - **Parameterized systems** : System sizing, Open Systems

Question

How to verify these systems ?

Context

Computer-aided proof adapted to π -calculus are very few.

- [Mobility Workbench](#) (Victor and Moller, 1994) : Open bisimulation equivalent and state space generation "on-the-fly"
- [HAL](#) (Ferrari et al., 2003) : historical substitutions of names with HD-Automa (HAL)

Context

Computer-aided proof adapted to π -calculus are very few.

- **Mobility Workbench** (Victor and Moller, 1994) : Open bisimulation equivalent and state space generation "on-the-fly"
- **HAL** (Ferrari et al., 2003) : historical substitutions of names with HD-Automa (HAL)

Scope of our work

We want to translate these systems into networks of timed automata to use UPPAAL toolkit (a more friendly tool!).

Outline

- 1 Introduction
- 2 From π -calculus terms to timed automata
- 3 A case study : MCA Architecture
- 4 Conclusion and future works

UPPAAL toolkit

- Integrated tool environment for **modeling**, validation and **verification** of real-time systems
- Systems modeled as **networks of timed automata**
- The Uppaal modeling language extends timed automata with the following **additional features** :
 - automata are defined with a set of parameters
 - committed locations
 - arrays (clocks, channels, constants, integers)
 - broadcast channels
 - ...

A Short Example

- We consider 2 terms written in π -calculus

$$P(x) \stackrel{\text{def}}{=} (\nu u) (\bar{x}\langle u \rangle . u(a) . 0) \text{ and } Q(z) \stackrel{\text{def}}{=} z(b) . (\nu c) \bar{b}\langle c \rangle . 0$$

A Short Example

- We consider 2 terms written in π -calculus

$$P(x) \stackrel{\text{def}}{=} (\nu u) (\bar{x}\langle u \rangle . u(a) . 0) \text{ and } Q(z) \stackrel{\text{def}}{=} z(b) . (\nu c) \bar{b}\langle c \rangle . 0$$

- which the successive reductions lead to define 3 states

$$P : P \xrightarrow{\bar{x}\langle u \rangle} P' \xrightarrow{u(a)} 0$$

$$Q : Q \xrightarrow{z(b)} Q' \xrightarrow{\bar{b}\langle c \rangle} 0$$

A Short Example

- We consider 2 terms written in π -calculus

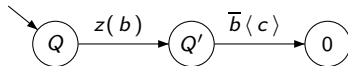
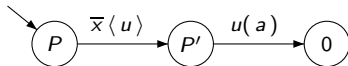
$$P(x) \stackrel{\text{def}}{=} (\nu u) (\bar{x}\langle u \rangle . u(a) . 0) \text{ and } Q(z) \stackrel{\text{def}}{=} z(b) . (\nu c) \bar{b}\langle c \rangle . 0$$

- which the successive reductions lead to define 3 states

$$P : P \xrightarrow{\bar{x}\langle u \rangle} P' \xrightarrow{u(a)} 0$$

$$Q : Q \xrightarrow{z(b)} Q' \xrightarrow{\bar{b}\langle c \rangle} 0$$

- A translation into 2 automata might be



A Short Example

- We consider 2 terms written in π -calculus

$$P(x) \stackrel{\text{def}}{=} (\nu u) (\bar{x}\langle u \rangle . u(a) . 0) \text{ and } Q(z) \stackrel{\text{def}}{=} z(b) . (\nu c) \bar{b}\langle c \rangle . 0$$

- which the successive reductions lead to define 3 states

$$P : P \xrightarrow{\bar{x}\langle u \rangle} P' \xrightarrow{u(a)} 0$$

$$Q : Q \xrightarrow{z(b)} Q' \xrightarrow{\bar{b}\langle c \rangle} 0$$

- A translation into 2 automata might be



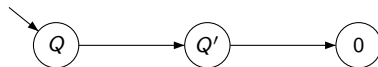
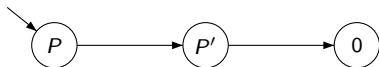
⚠ We must define a execution context ⚠

A Short Example

We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right)$$

$$\stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . 0 \right) \mid x(b) . (\nu c) \bar{b}\langle c \rangle . 0$$

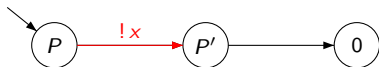


A Short Example

We consider a term *System* such as

$$\begin{aligned} \text{System} &\stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \\ &\stackrel{\text{def}}{=} (\nu u) \left(\bar{x}(u) . u(a) . 0 \right) \mid x(b) . (\nu c) \bar{b}(c) . 0 \end{aligned}$$

① Communication on x



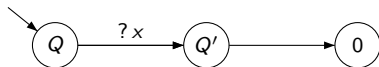
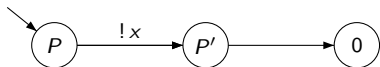
A Short Example

We consider a term *System* such as

$$\begin{aligned} \text{System} &\stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \\ &\stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . 0 \right) \mid x(b) . (\nu c) \bar{b}\langle c \rangle . 0 \end{aligned}$$

- 1 Communication on x
- 2 Reduction

$$\text{System} \xrightarrow{\tau} (\nu u) \left(u(a) . 0 \mid (\nu c) \bar{u}\langle c \rangle . 0 \right)$$



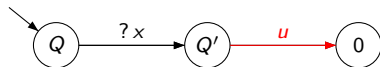
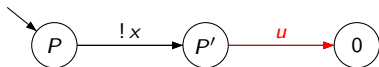
A Short Example

We consider a term *System* such as

$$\begin{aligned} \text{System} &\stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \\ &\stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . 0 \right) \mid x(b) . (\nu c) \bar{b}\langle c \rangle . 0 \end{aligned}$$

- 1 Communication on x
- 2 Reduction
- 3 Scope extrusion

$$\text{System} \xrightarrow{\tau} (\nu u) \left(u(a) . 0 \mid (\nu c) \bar{u}\langle c \rangle . 0 \right)$$



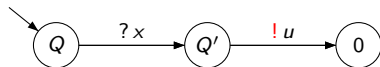
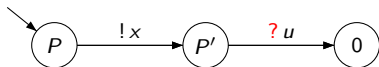
A Short Example

We consider a term *System* such as

$$\begin{aligned} \text{System} &\stackrel{\text{def}}{=} (\nu x) \left(P(x) \mid Q(x) \right) \\ &\stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . 0 \mid x(b) . (\nu c) \bar{b}\langle c \rangle . 0 \right) \end{aligned}$$

- 1 Communication on x
- 2 Reduction
- 3 Scope extrusion
- 4 Communication on u

$$\text{System} \xrightarrow{\tau} (\nu u) \left(u(a) . 0 \mid (\nu c) \bar{u}\langle c \rangle . 0 \right)$$



Operational Semantics

Formally, a automaton $\mathcal{P}(id_P)$, translation of a π -calculus term P , is a automaton such as :

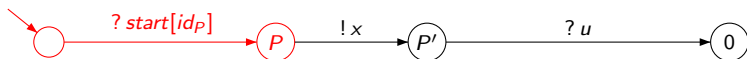
- id_P is a **unique identifier**



Operational Semantics

Formally, a automaton $\mathcal{P}(id_P)$, translation of a π -calculus term P , is a automaton such as :

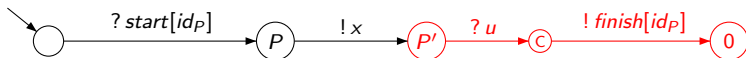
- id_P is a **unique identifier**
- if q_0 is the initial state then $\langle q_0, ? start[id_P], P \rangle$ is the **first transition**



Operational Semantics

Formally, a automaton $\mathcal{P}(id_P)$, translation of a π -calculus term P , is a automaton such as :

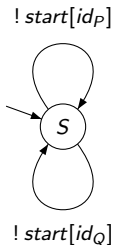
- id_P is a **unique identifier**
- if q_0 is the initial state then $\langle q_0, ? start[id_P], P \rangle$ is the **first transition**
- if a term Q waits the end of P execution and $P' \xrightarrow{\alpha} 0$ then the double transition $\langle P', \alpha, ! finish[id_P], 0 \rangle$ is the last of \mathcal{P} and Q , translation of Q , have a transition $\langle Q', ? finish[id_P], Q'' \rangle$.



Operational Semantics

A special case : \mathcal{S} , automaton representing the term *System*, has neither ID nor the labeled transition ? *start*[...].

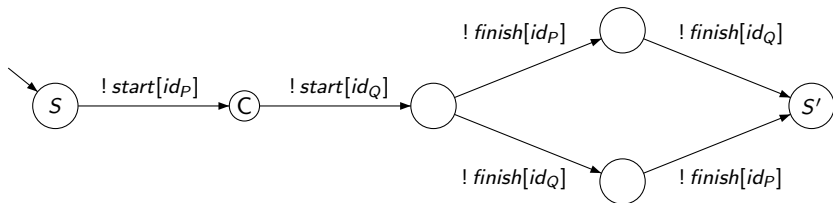
\mathcal{S} , translation of *System* $\stackrel{def}{=} (P \mid Q)$



Operational Semantics

A special case : \mathcal{S} , automaton representing the term *System*, has neither ID nor the labeled transition ? *start*[...].

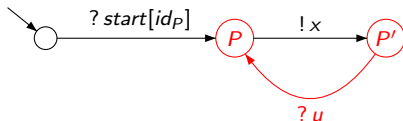
\mathcal{S} , translation of *System* $\stackrel{\text{def}}{=} (P \mid Q) . S'$



Operational Semantics

Translations of some π -calculus operators

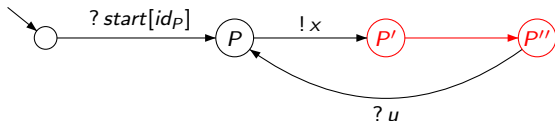
- **Recursion** : $P(x) \stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . P(x) \right)$



Operational Semantics

Translations of some π -calculus operators

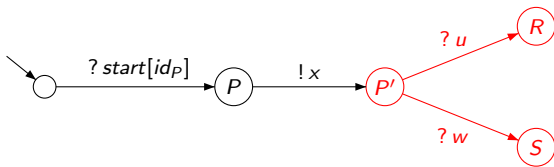
- **Recursion** : $P(x) \stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . P(x) \right)$
- **Silent action** : $P(x) \stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . \tau . u(a) . P(x) \right)$



Operational Semantics

Translations of some π -calculus operators

- **Recursion** : $P(x) \stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . u(a) . P(x) \right)$
- **Silent action** : $P(x) \stackrel{\text{def}}{=} (\nu u) \left(\bar{x}\langle u \rangle . \tau . u(a) . P(x) \right)$
- **Sum** : $P(x) \stackrel{\text{def}}{=} (\nu u, w) \left(\bar{x}\langle u, w \rangle . \left(u(a) . R + w(a) . T \right) \right)$



Mobility

- We consider a term *System* such as

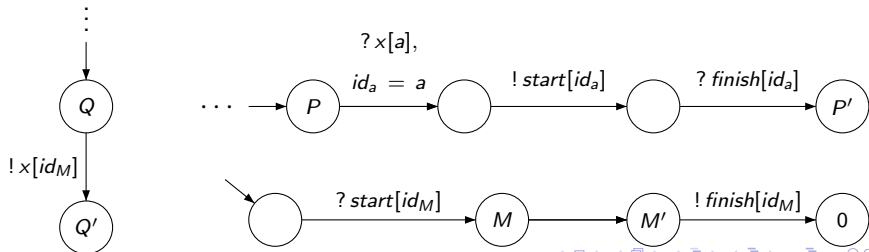
$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \left\{ \begin{array}{l} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{array} \right.$$

Mobility

- We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \begin{cases} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

- This system is translated into a network of automata such as

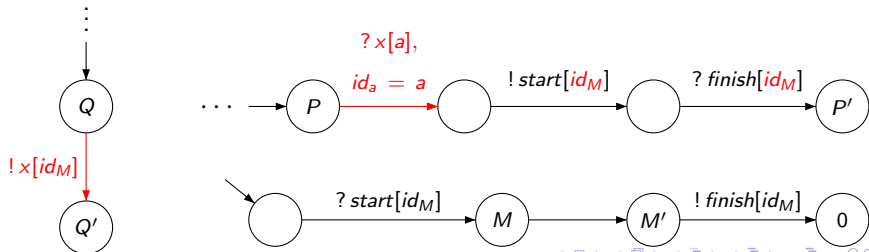


Mobility

- We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \left\{ \begin{array}{l} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{array} \right.$$

- This system is translated into a network of automata such as

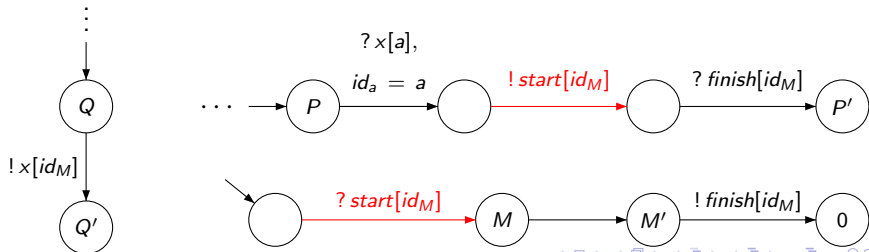


Mobility

- We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \begin{cases} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

- This system is translated into a network of automata such as

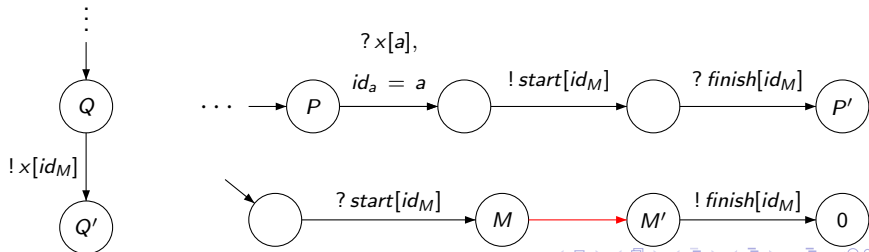


Mobility

- We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \begin{cases} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

- This system is translated into a network of automata such as

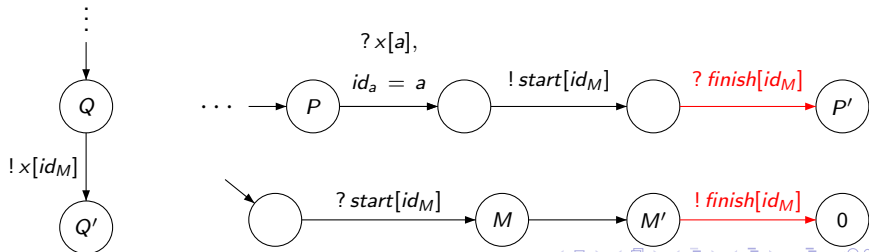


Mobility

- We consider a term *System* such as

$$\text{System} \stackrel{\text{def}}{=} (\nu x) (P(x) \mid Q(x)) \begin{cases} M \stackrel{\text{def}}{=} \tau . M' . 0 \\ Q(z) \stackrel{\text{def}}{=} \bar{z} \langle M \rangle . Q' \\ P(x) \stackrel{\text{def}}{=} x(A) . A . P' \end{cases}$$

- This system is translated into a network of automata such as



Time for fault tolerance

- In Mobile Agent System, like any distributed system, errors or accidents are inevitable.
 - Design errors, programming errors, ...
 - Environmental accidents (network, hardware, ...)
 - ...
- A Mobile Agent System must be a **fault-tolerant system**.
- To model a fault-tolerant system, we add a **notion of time** to our automata with **Timed Automata**.

Outline

- 1 Introduction
- 2 From π -calculus terms to timed automata
- 3 A case study : MCA Architecture**
 - Presentation
 - Modeling
 - Verification
- 4 Conclusion and future works

MCA Architecture

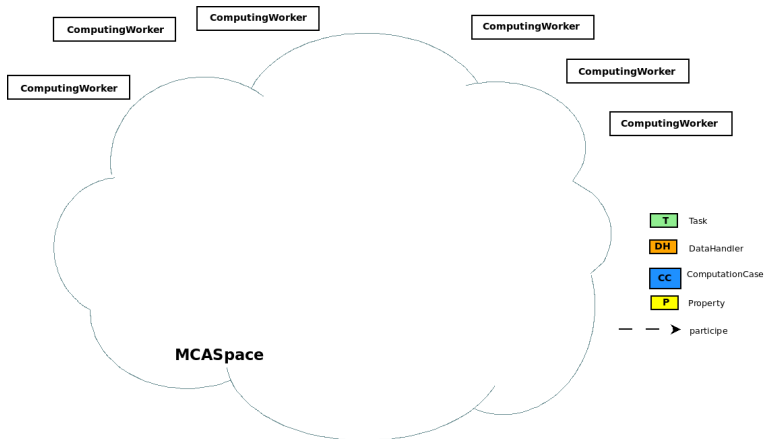
MCA (Mobile Computing Architecture) is :

- A platform for the resolution of cases of numerical computation
- Capable to execute algorithms implemented in Java or in native language thanks to LLVM¹ (Low Level Virtual Machine)
- Implemented in Java
- Based on a JavaSpace (provided by Jini API)

1. <http://llvm.org/>

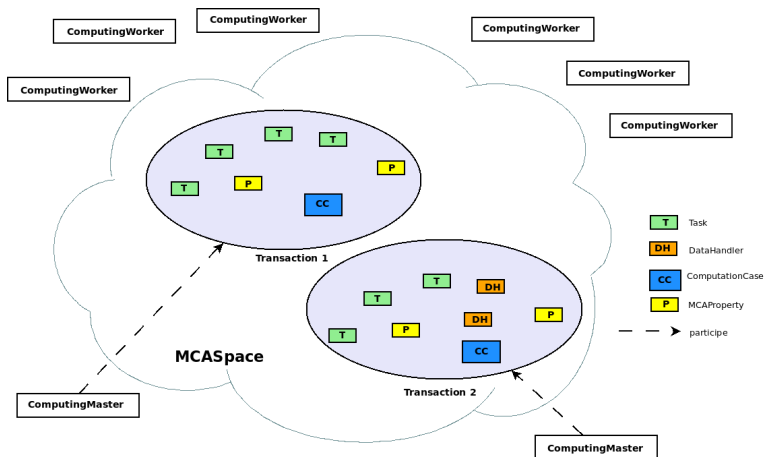
MCA Architecture

Key components



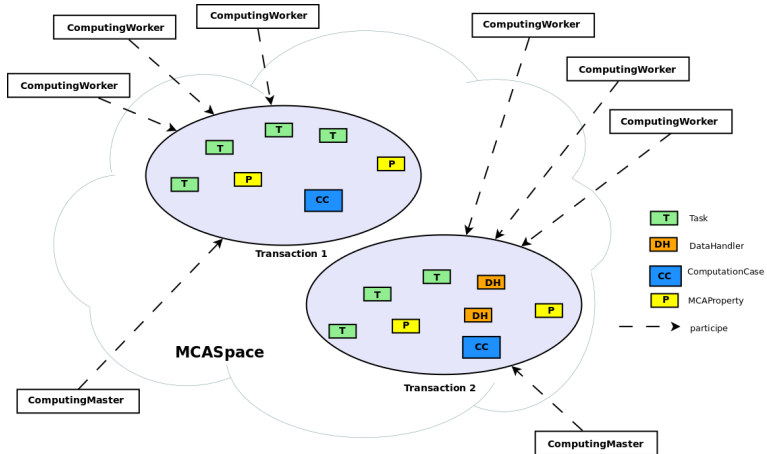
MCA Architecture

Key components



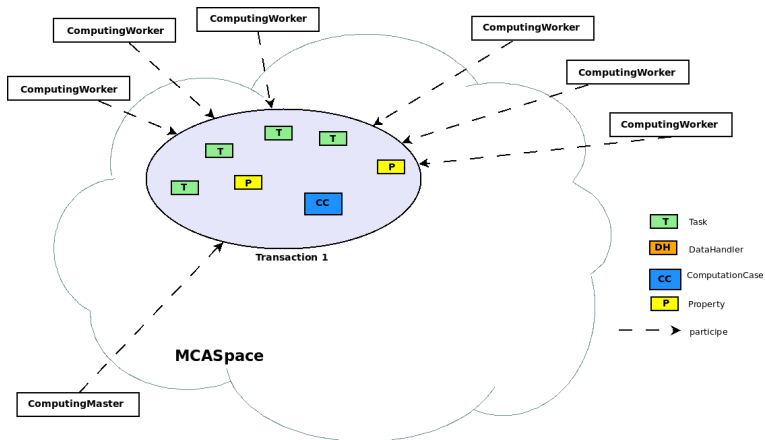
MCA Architecture

Key components



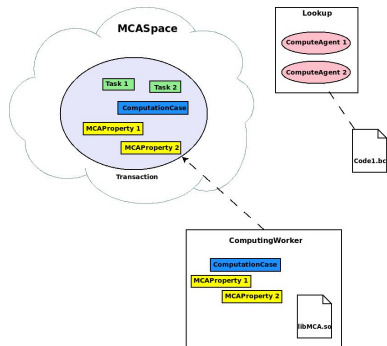
MCA Architecture

Key components



MCA Architecture

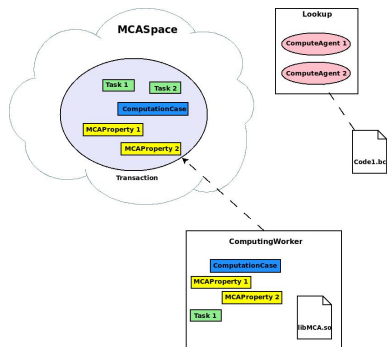
Task execution



MCA Architecture

Task execution

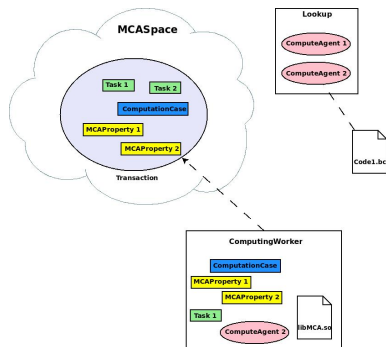
- 1 *CW* takes a *Task* object which contains the *ComputeAgent* name to execute



MCA Architecture

Task execution

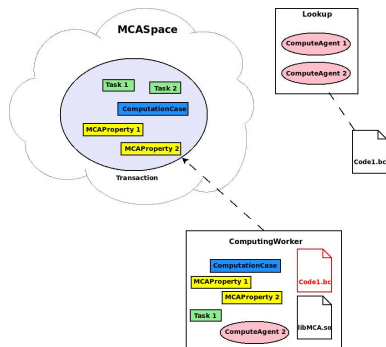
- 1 CW takes a *Task* object which contains the *ComputeAgent* name to execute
- 2 CW gets this *ComputeAgent* registered on a Jini Lookup



MCA Architecture

Task execution

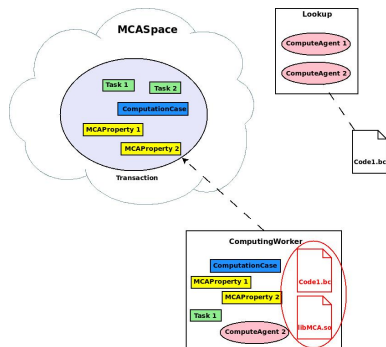
- 1 CW takes a *Task* object which contains the *ComputeAgent* name to execute
- 2 CW gets this *ComputeAgent* registered on a Jini Lookup
- 3 CW downloads associated LLVM bytecode file



MCA Architecture

Task execution

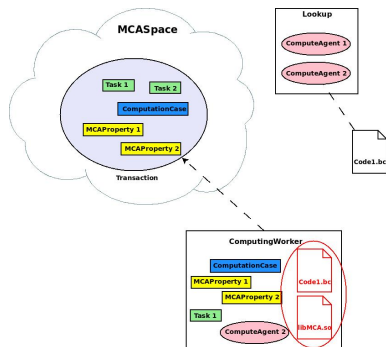
- 1 CW takes a *Task* object which contains the *ComputeAgent* name to execute
- 2 CW gets this *ComputeAgent* registered on a Jini Lookup
- 3 CW downloads associated LLVM bytecode file
- 4 CW loads *libMCA* library and executes the native function with [JNI API](#)



MCA Architecture

Task execution

- 1 CW takes a *Task* object which contains the *ComputeAgent* name to execute
- 2 CW gets this *ComputeAgent* registered on a *Jini Lookup*
- 3 CW downloads associated **LLVM bytecode file**
- 4 CW loads *libMCA* library and executes the native function with **JNI API**
- 5 CW updates the *Task* state on *MCASpace*



MCA Modeling

A system to solve a computation case consists of

- 1 workspace

which translates into a network of timed automata such as

$$\mathcal{M}_{space}(id_s) \parallel \mathcal{T}_{dir}(id_{td})$$

MCA Modeling

A system to solve a computation case consists of

- 1 workspace
- 1 computation case

which translates into a network of timed automata such as

$$\mathcal{M}_{space}(id_s) \parallel \mathcal{T}_{dir}(id_{td}) \parallel \mathcal{M}(id_m) \parallel \mathcal{C}c(id_{cc})$$

MCA Modeling

A system to solve a computation case consists of

- 1 workspace
- 1 computation case
- n tasks and their respective mobile agents

which translates into a network of timed automata such as

$$\mathcal{M}_{space}(id_s) \parallel \mathcal{T}_{dir}(id_{td}) \parallel \mathcal{M}(id_m) \parallel \mathcal{C}c(id_{cc}) \\ \parallel \mathcal{T}(id_{t_1}) \parallel \dots \parallel \mathcal{T}(id_{t_n}) \parallel \mathcal{C}a(id_{c_1}) \parallel \dots \parallel \mathcal{C}a(id_{c_n})$$

MCA Modeling

A system to solve a computation case consists of

- 1 workspace
- 1 computation case
- n tasks and their respective mobile agents
- m Worker agents

which translates into a network of timed automata such as

$$\begin{aligned}
 & \mathcal{M}_{space}(id_s) \parallel \mathcal{T}_{dir}(id_{td}) \parallel \mathcal{M}(id_m) \parallel \mathcal{C}c(id_{cc}) \\
 & \parallel \mathcal{T}(id_{t_1}) \parallel \cdots \parallel \mathcal{T}(id_{t_n}) \parallel \mathcal{C}a(id_{c_1}) \parallel \cdots \parallel \mathcal{C}a(id_{c_n}) \\
 & \parallel \mathcal{W}(id_{w_1}) \parallel \cdots \parallel \mathcal{W}(id_{w_m})
 \end{aligned}$$

MCA Properties

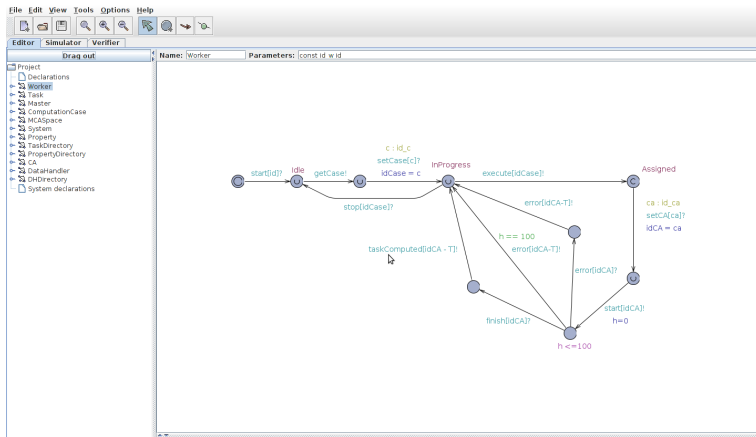
Using our network of timed automata, we can verify properties specific to our architecture based on a system of mobile agents.

We expressed the following properties :

- To ensure that a task put on the *MCASpace* is necessarily executed if a *Worker* agent is available
- To assert that a task are never accessed by two *Worker* agents at same time
- To assert that each *Worker* agent is given a chance to execute a task.
- To assert that the system will never get itself into deadlock
- ...

Modeling with UPPAAL

Modeling



Modeling with UPPAAL

Simulation

The screenshot displays the UPPAAL simulation environment. The main window shows a hierarchical view of the system components, including:

- System** (root)
- MCASpace(1)**
- TaskDirectory(2)**
- PropertyDirectory(3)**
- ComputationCase(11)**
- Master(10)**
- Worker(14)**
- Property(12)**
- Property(13)**
- Worker(15)**
- Task(16)**
- Task(17)**
- CA(18)**
- CA(19)**

Each component is represented by a state transition graph. The left sidebar contains the **Editor** and **Simulator** tabs. The **Simulator** tab shows the **Enabled Transitions** and **Simulation Trace**. The **Simulation Trace** shows a sequence of events:

```

h >= 0
Worker(14).h
Worker(15).h
Task(16).h >=
Task(17).h >=
CA(18).cTime
CA(19).cTime
h = Worker(1
Worker(14).h
Worker(15).h
Task(16).h =
Task(17).h =
CA(18).cTime
CA(19).cTime

```

At the bottom, there is a **Trace File** section with buttons for **Prev**, **Next**, **Replay**, **Open**, **Save**, and **Auto**, along with a speed control slider from **Slow** to **Fast**.

Modeling with UPPAAL

Verification

The screenshot shows the UPPAAL verification tool interface. The main window is divided into several sections:

- Overview:** A list of events and their status. The first event, "A[] deadlock imply Master(idMaster1).Finished", is highlighted in blue. To its right are three colored circles: red, red, and green. Below the list are buttons for "Check", "Insert", "Remove", and "Comments".
- Query:** A text area containing the query "E<=> Master(10).Finished".
- Comment:** An empty text area for user comments.
- Status:** A scrollable log showing the execution history. It includes messages like "Established direct connection to local server.", "(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server. Disconnected.", and the final result: "A[] deadlock imply Master(idMaster1).Finished. Property is not satisfied." followed by a list of events with their corresponding property satisfaction status (e.g., "E<=> Master(10).Error. Property is satisfied.", "Task(16).Added -> Task(16).Finished. Property is not satisfied.", etc.).

At the bottom of the window, there is a toolbar with navigation icons (back, forward, search, etc.) and a status bar.

Outline

- 1 Introduction
- 2 From π -calculus terms to timed automata
- 3 A case study : MCA Architecture
- 4 Conclusion and future works

Conclusion

Conclusion

- Modeling of a Mobile Agent System into $HO\pi$ -calculus terms
- Translation of $HO\pi$ -calculus terms into timed automata
- Model-Checking of our MCA Architecture with UPPAAL toolkit

Future works

- Manage code generation with the use of formal specification
 - Define new operator to get a skeleton of mobile code
 - Keep initial assertions into programming definition

Questions ?