Verification of a Management Protocol for Cloud Applications

Gwen Salaün

Grenoble INP, Inria, France

joint work with

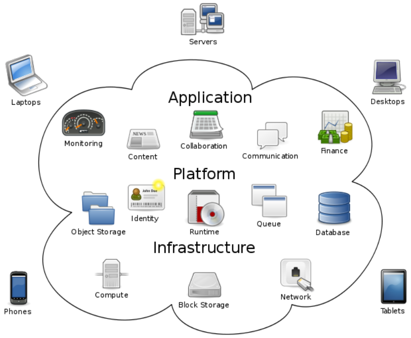
Rim Abid[1,2], Francesco Bongiovanni[2], Noel De Palma[2]

[1]Inria, Grenoble, France

[2]UJF-Grenoble 1, France

Introduction

- Cloud computing aims at delivering resources and applications as a service over a network (*e.g.*, the Internet)



Cloud Computing

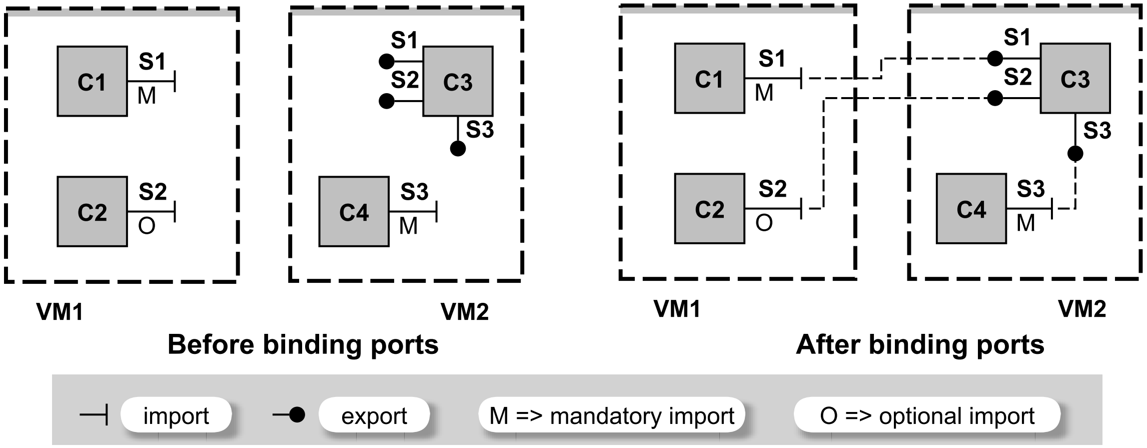
- Cloud applications are often complex distributed applications composed of multiple software running on separate virtual machines
- Setting up, (re)configuring, and monitoring these applications are difficult tasks, and involve complex management protocols
- In this talk, we present the verification of an innovative reconfiguration protocol, which automates the management of cloud applications running over several virtual machines

Outline

1. Reconfiguration Protocol
2. LNT and CADP
3. Specification in LNT
4. Verification with CADP
5. Concluding Remarks

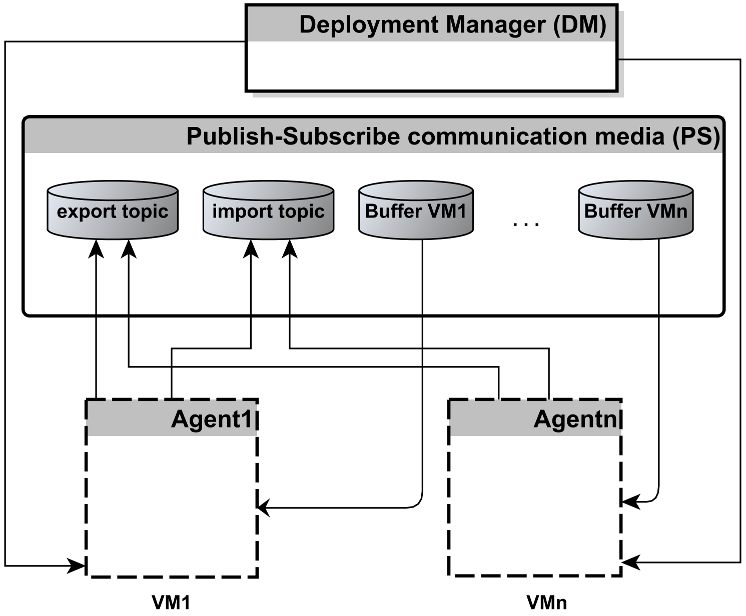
Application Model

- An application model consists of a set of components distributed over several virtual machines
- Each component requires or provides services through imports (optional or mandatory) and exports, respectively
- Ports are typed and match when they share the same type
- Bindings connect one import to one export with the same type, locally (same VM) or remotely



Participants

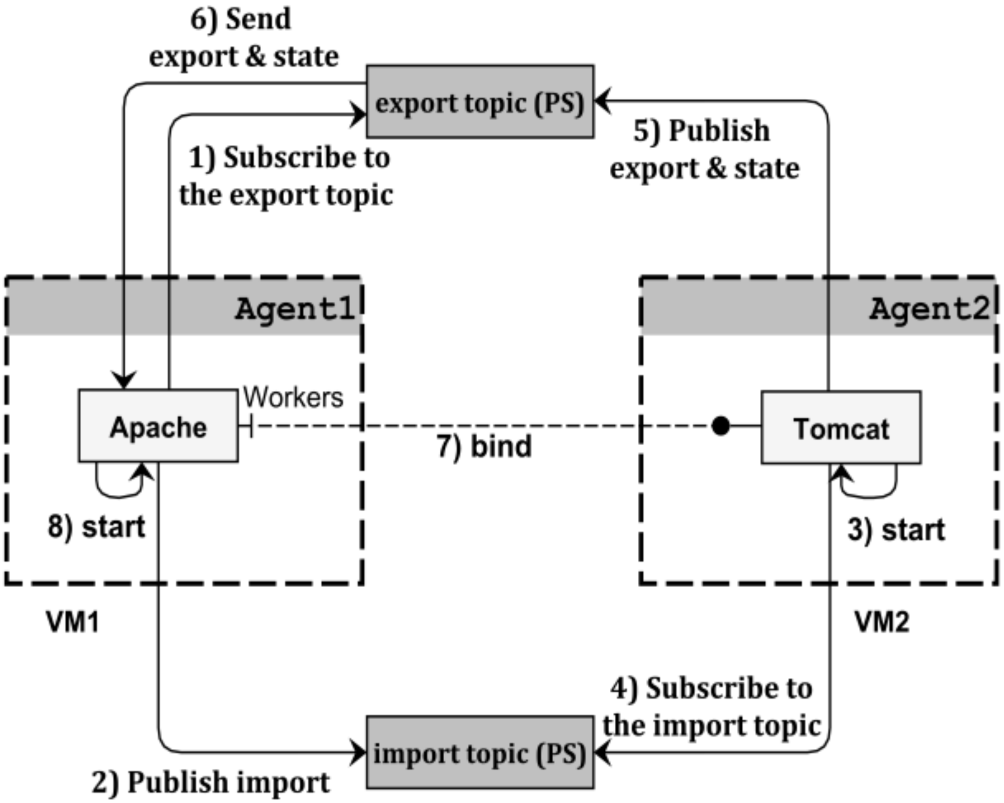
- The deployment manager (DM) guides the reconfiguration by instantiating/destroying VMs
- Each VM is equipped with a configuration agent in charge of (dis) connecting ports and starting/stopping components
- Communications between DM/VM and VMs are handled by a publish-subscribe (PS) messaging system



VM Instantiation (1/2)

- When a VM is instantiated, the agent is in charge of starting all the components
- A component without imports or optional imports only can be started immediately
- Otherwise, each mandatory import requires an export with the same type
- The PS is used to resolve compatible dependencies and exchange start-up information
- A component can be started when all its mandatory imports are bound to started components

VM Instantiation (2/2)



Instantiating VM1 then Instantiating VM2

VM Destruction (1/2)

- All components on a VM to be destroyed need to be properly stopped as well as all components bound on them through mandatory imports

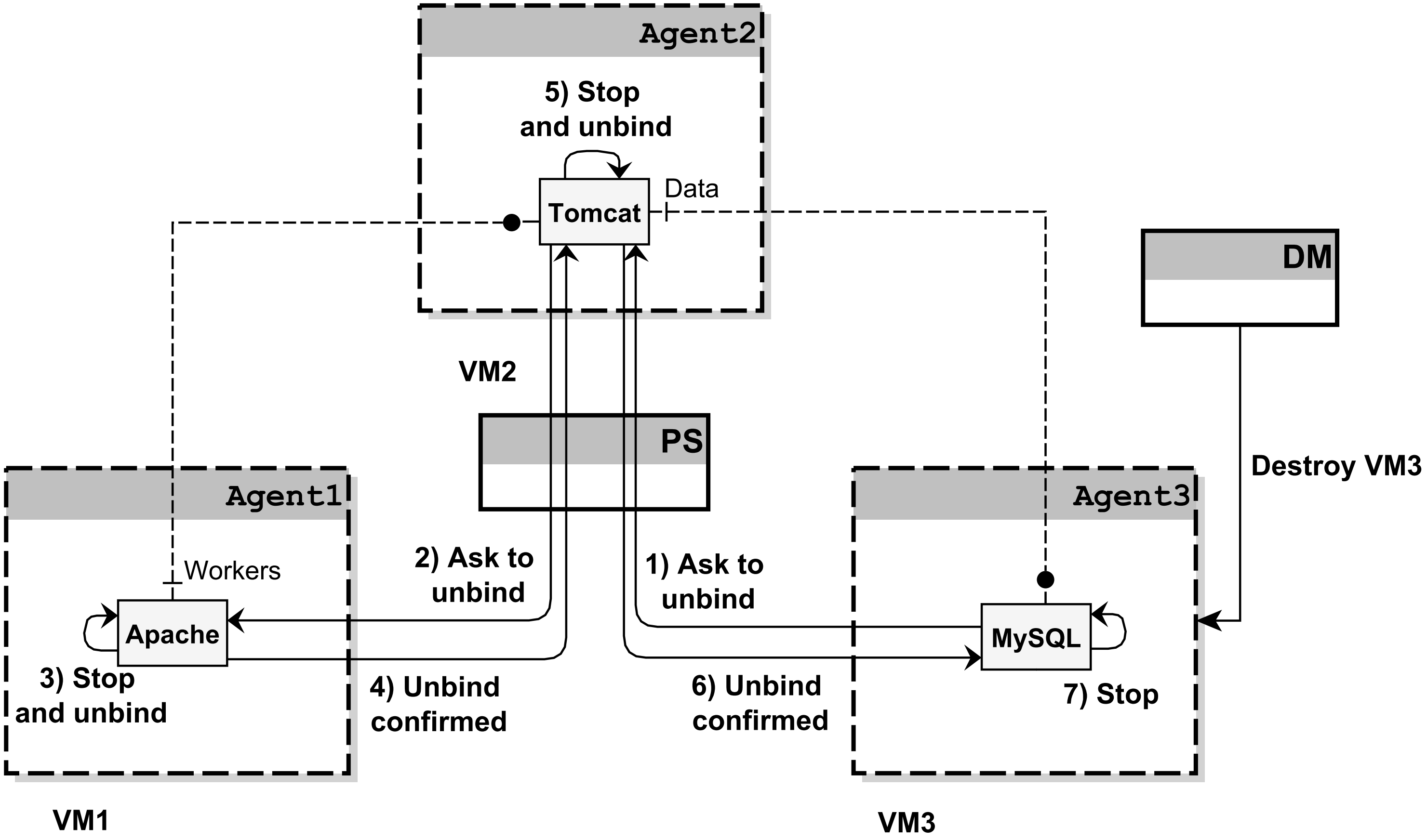
- A component that does not provide any service can be immediately stopped

- Shutting down a component implies a backward propagation of "*ask to unbind*" messages via the PS

- A forward propagation of "unbind confirmed" messages lets the components know that disconnection has been achieved

- When a component has received such messages for all its mandatory imports, it can stop itself

VM Destruction (2/2)



Outline

1. Reconfiguration Protocol
2. LNT and CADP
3. Specification in LNT
4. Verification with CADP
5. Concluding Remarks

LOTOS NT

- LOTOS NT (LNT) is a value-passing process algebra with user-friendly syntax and operational semantics

- LNT is an imperative-like language where you can specify data types, functions (pattern matching and recursion), and processes

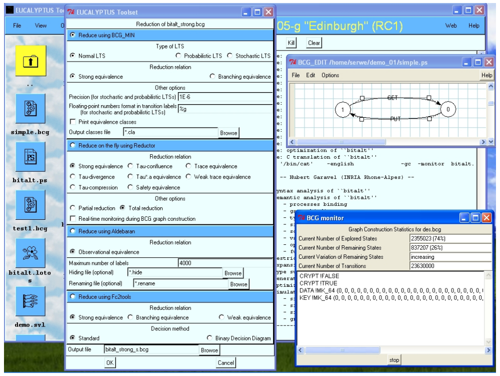
- Excerpt of the LNT process grammar:

B ::= **stop** | G(!E, ?X) **where** E' | **if** E **then** B1 **else** B2 **end if**
| x:=E | **hide** G **in** B **end hide** | P [G1,...,Gm] (E1,...,En)
| **select** B1 **[]** … **[]** Bn **end select** | B1 **;** B2
| **par** G **in** B1 **||** … **||** Bn **end par**

- Verification using CADP through an automated translation to LOTOS

Construction and Analysis of Distributed Processes (CADP)

- Design of asynchronous systems
  - Concurrent processes
  - Message-passing communication
  - Nondeterministic behaviours



EUCALYPTUS Toolset

Reduction of bitalt_strong.bcg

● Reduce using BCG_MIN

Type of LTS

● Normal LTS ○ Probabilistic LTS ○ Stochastic LTS

Reduction relation

● Strong equivalence ○ Branching equivalence

Other options

Precision (for stochastic and probabilistic LTSs) 1E-6

Floating-point numbers format in transition labels (for stochastic and probabilistic LTSs) %g

□ Print equivalence classes

Output classes file *.cla Browse

○ Reduce on the fly using Reductor

Reduction relation

● Strong equivalence ○ Tau-confluence ○ Trace equivalence

○ Tau-divergence ○ Tau*.a equivalence ○ Weak trace equivalence

○ Tau-compression ○ Safety equivalence

Other options

○ Partial reduction ● Total reduction

□ Real-time monitoring during BCG graph construction

○ Reduce using Aldebaran

Reduction relation

● Observational equivalence

Maximum number of labels 4000

Hiding file (optional) *.hide Browse

Renaming file (optional) *.rename Browse

○ Reduce using Fc2tools

Reduction relation

● Strong equivalence ○ Branching equivalence ○ Weak equivalence

Decision method

● Standard ○ Binary Decision Diagram

Output file bitalt_strong_s.bcg Browse

OK Cancel

05-g "Edinburgh" (RC1)

```
e:  optimization of ``bitalt''
e:  C translation of ``bitalt''
 '/bin/cat'    -english              -gc  -monitor  bitalt.

 -- Hubert Garavel (INRIA Rhone-Alpes) --

yntax analysis of ``bitalt''
emantic analysis of ``bitalt''
  - processes binding
```

BCG_EDIT /home/serwe/demo_01/simple.ps

BCG monitor

Graph Construction Statistics for des.bcg

Current Number of Explored States	2355023 (74%)
Current Number of Remaining States	837207 (26%)
Current Variation of Remaining States	increasing
Current Number of Transitions	23630000

```
CRYPT !FALSE
CRYPT !TRUE
DATA !MK_64 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
KEY !MK_64 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

- Formal approach rooted in concurrency theory: process calculi, Labeled Transition Systems, bisimulations, branching temporal logics

- Many verification techniques: simulation, model and equivalence checking, compositional verification, test case generation, performance evaluation, etc.

- Numerous real-world applications: avionics, embedded systems, hardware design, middleware and software architectures, etc.

Outline

1. Reconfiguration Protocol
2. LNT and CADP
3. Specification in LNT
4. Verification with CADP
5. Concluding Remarks

Specification in LNT (1/2)

- The specification consists of three parts: data types (200 lines), functions (800 lines), processes (1200 lines)
- Data types describe the application model (VMs, components, ports) and the communication model (messages, buffers, topics)
- Functions apply on to data expressions for, *e.g.*, extracting information from the application model or adding/retrieving messages from buffers

```
function add (m: TMessage, q: TBuffer): TBuffer is
        case q in
        var hd: TMessage, tl: TBuffer in
                nil -> return cons(m,nil)
        |       cons(hd,tl) -> return cons(hd,add(m,tl))
        end case
end function
```

Specification in LNT (2/2)

- Processes specify the participants of the protocol: the deployment manager, the PS messaging system, and one agent per VM
- Actions correspond either to interactions between processes or specific moments of the protocol execution (useful for verification purposes)

```
par INSTANTIATEVM, DESTROYVM in
      DM [INSTANTIATEVM, DESTROYVM] (appli)
||
      par AGENTtoPS1, PStoAGENT1, ... in
            par
                 Agent [INSTANTIATEVM, AGENTtoPS1, PStoAGENT1, DESTROYVM,
                   STARTCOMPO, BINDCOMPO, STOPCOMPO, UNBINDCOMPO]  (vm1)
            ||
                 Agent [...] (vm2)
            end par
      ||
            PS [AGENTtoPS1, ..., PStoAGENT2] (!?ListBuffers)
      end par
end par
```

Application involving two virtual machines

Outline

1. Reconfiguration Protocol
2. LNT and CADP
3. Specification in LNT
4. Verification with CADP
5. Concluding Remarks

Properties

- We identified and checked 35 safety and liveness properties that must be preserved by the protocol

- Properties were specified in the MCL language (mu-calculus) and verified using the Evaluator 4.0 model checker

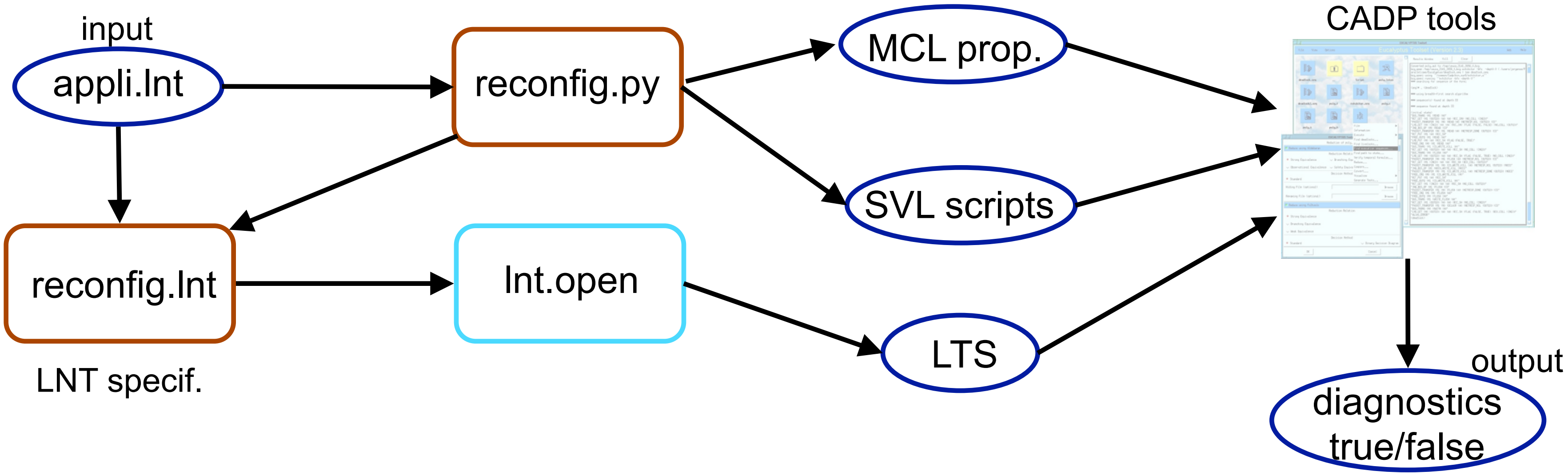
  - A component cannot be started before the components it depends on for mandatory imports

```
[ true* . “STARTCOMPO !Apache !VM1” .
  true* . “STARTCOMPO !Tomcat !VM2” ] false
```

  - A component hosted on a VM eventually stops after that VM receives a destruction request from the DM

```
( < true* . {DESTROYVM ?vm:String} .
    true* . {STOPCOMPO ?cid:String !vm} > true )
```

Verification



- Experiments were conducted on more than 600 hand-crafted examples (application model + reconfiguration scenario)

- Considering an application model with 4 VMs, 8 components, 7 imports to be bound, and 8 reconfiguration operations

  ⇒ the corresponding LTS consists of a few million states and transitions

  ⇒ the LTS generation and the verification of the 35 prop. takes a few hours

Problems Found

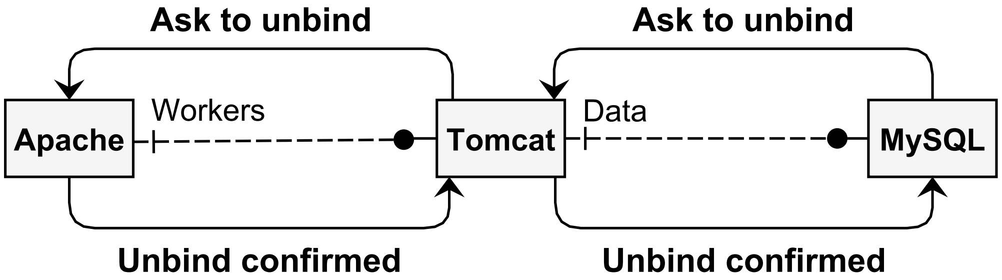
- Correction of several specific issues in the protocol, *e.g.*, adding some acknowledgement messages after effectively binding ports

- Replacing the component start-up/shutdown driven by the deployment manager with a distributed start-up/shutdown delegated to VM agents

  ⇒ reduction of the messages transmitted to and from the DM

- Detection of a bug in the VM destruction process thanks to a property stating that "*a component cannot be started and connected through an import to another component if that component is not started*"

  ⇒ corrected by stopping components in the right order



Outline

1. Reconfiguration Protocol
2. LNT and CADP
3. Specification in LNT
4. Verification with CADP
5. Concluding Remarks

Concluding Remarks

- We have presented the specification and verification of a reconfiguration protocol involving components distributed over several VMs
- The experience was successful because we detected several issues that were corrected in the corresponding Java implementation

Perspectives:

- Extension with finer-grained reconfiguration operations: addition and removal of components on already deployed VMs
- Extending the protocol to take VM failures into account: this implies restoring a consistent state for the application and possibly repairing it