# Fault-aware management of cloud-based applications

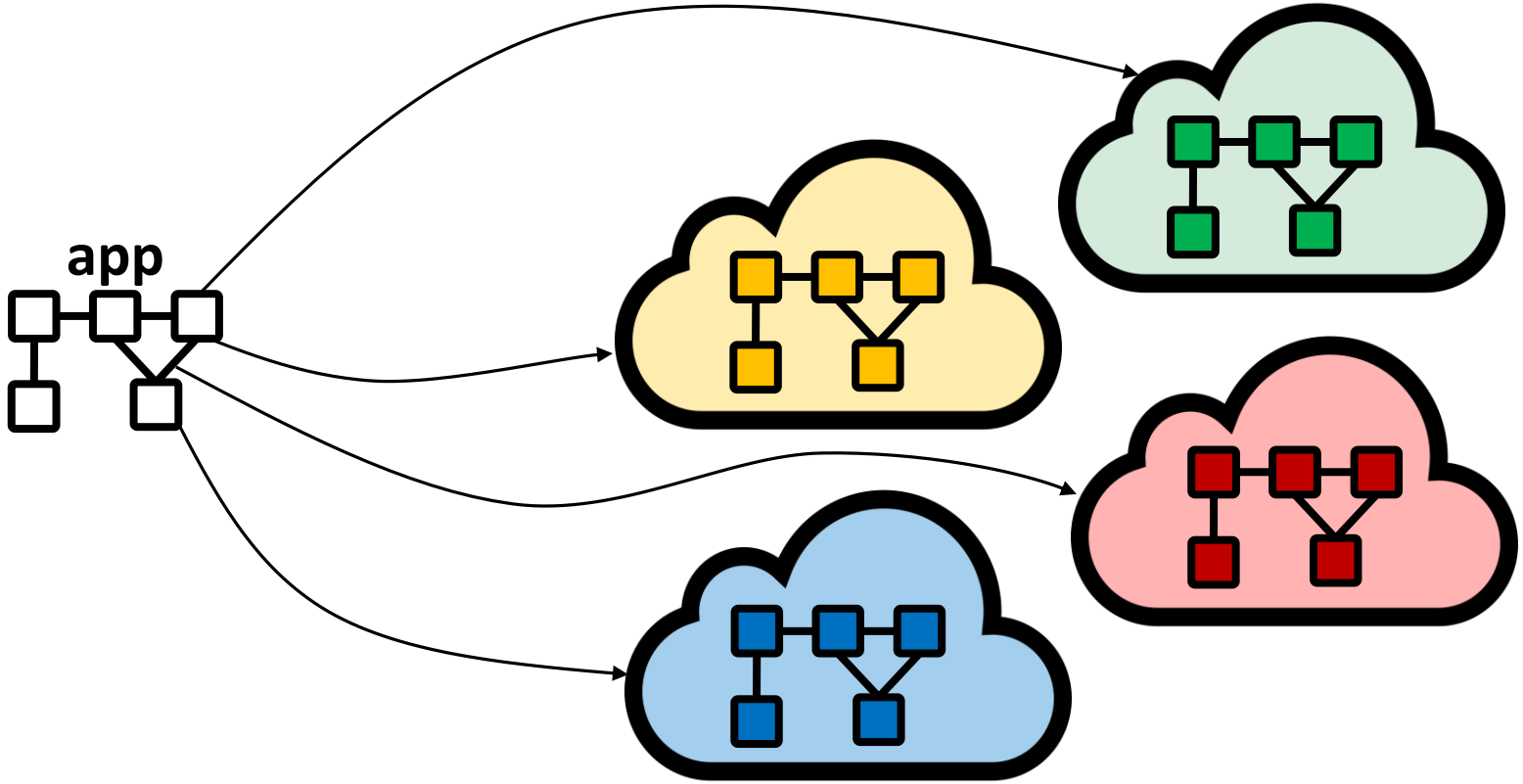
Jacopo Soldani

*Service, cloud and fog computing* seminars



https://di-unipi-socc.github.io

# Context



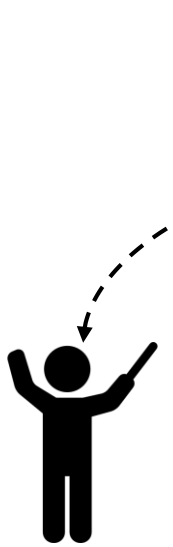

**Challenge[1]:**

- » Flexibly manage complex **composite applications**
- » over heterogeneous **cloud** platforms.

[1] F. Leymann. **Cloud computing**. it—Information Technology, 53(4):163–164, 2011.

# Two major issues[1,2]

Flexibly manage complex **composite applications** over heterogeneous **cloud** platforms.

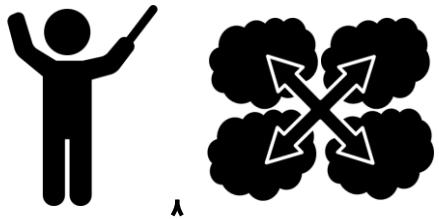


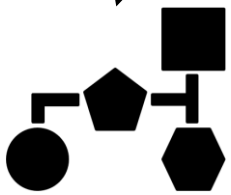
**Automate** the **management** of composite cloud applications.

**Support** a **vendor-agnostic design** of composite cloud applications.

[1] T. Binz et al. **TOSCA: Portable automated deployment and management of cloud applications**. Advanced Web Services, pp. 527-549, Springer, 2014.
[2] R. Di Cosmo et al. **Aeolus: A component model for the cloud**. Information and Computation, 239(0):100 –121, 2014.

# Our objectives





**Modelling** composite cloud applications.

**Compositional**, **fault-aware** modelling for the **management behaviour** of comp. cloud applications.

**Analysing** composite cloud applications.

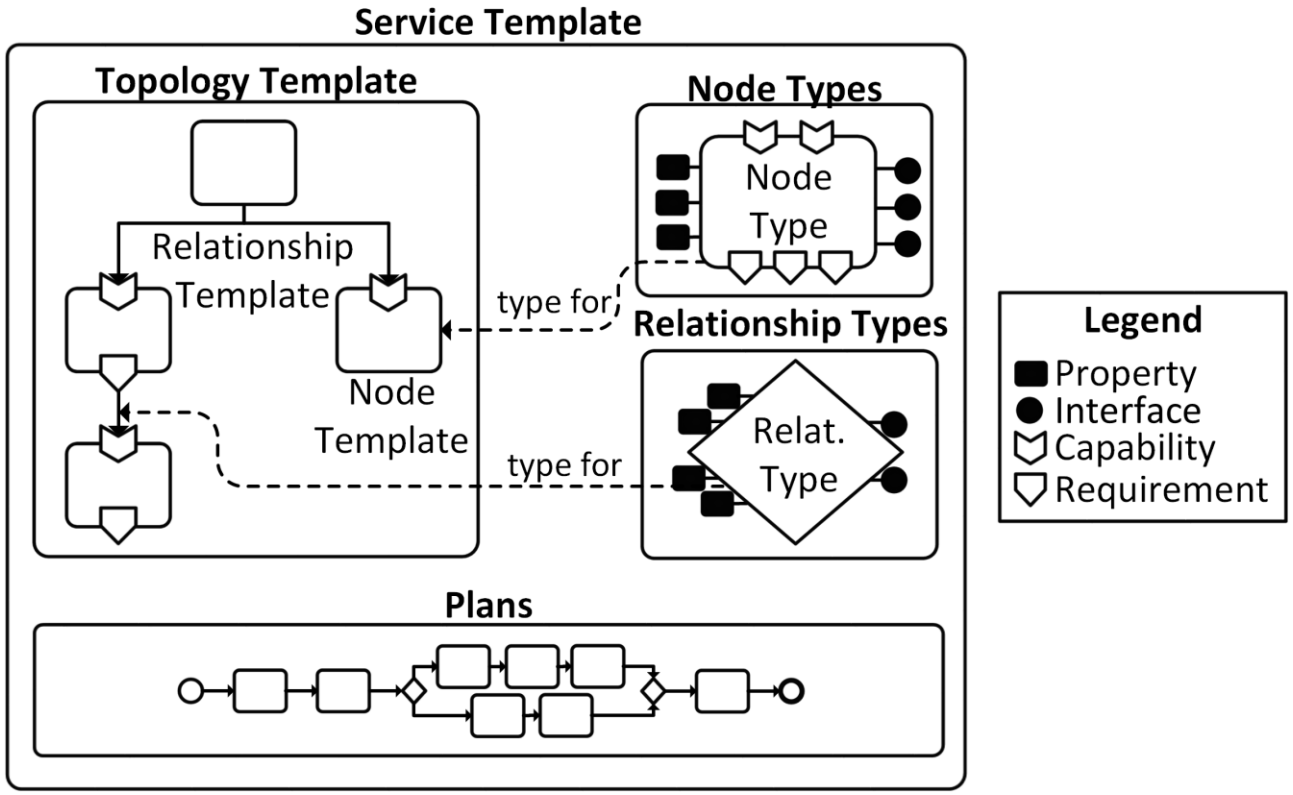
Techniques for **checking** and **planning** the **management** of comp. cloud applications.

# Outline

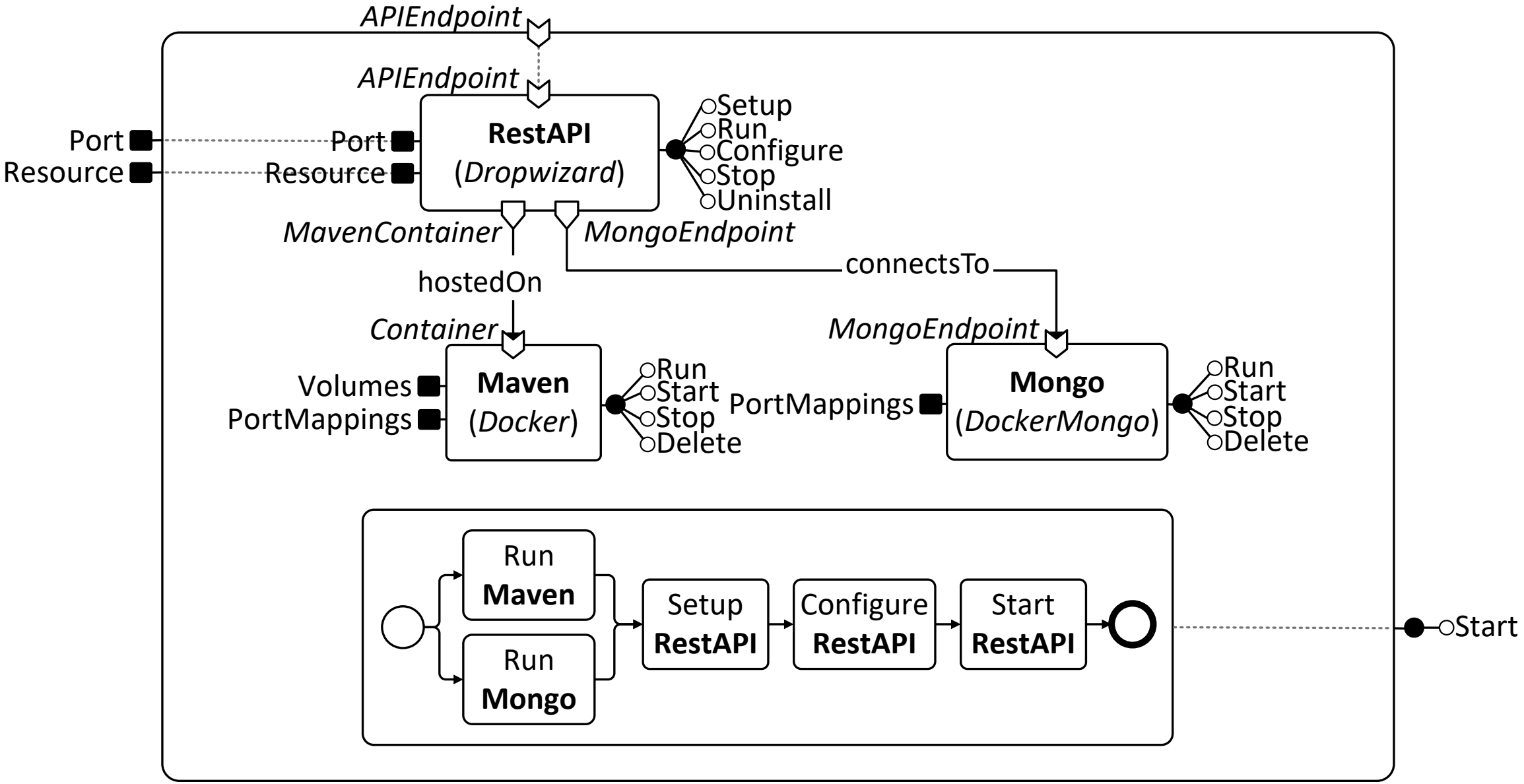
- The OASIS TOSCA standard
- Modelling and analysing cloud application management
- Fault-aware application management protocols
- Conclusions

# TOSCA (*Topology and Orchestration Specification for Cloud Applications*)

- OASIS **standard**
- **Goals**:
  1. Create portable cloud applications.
  2. Automate application management.



# A toy example



# Outline

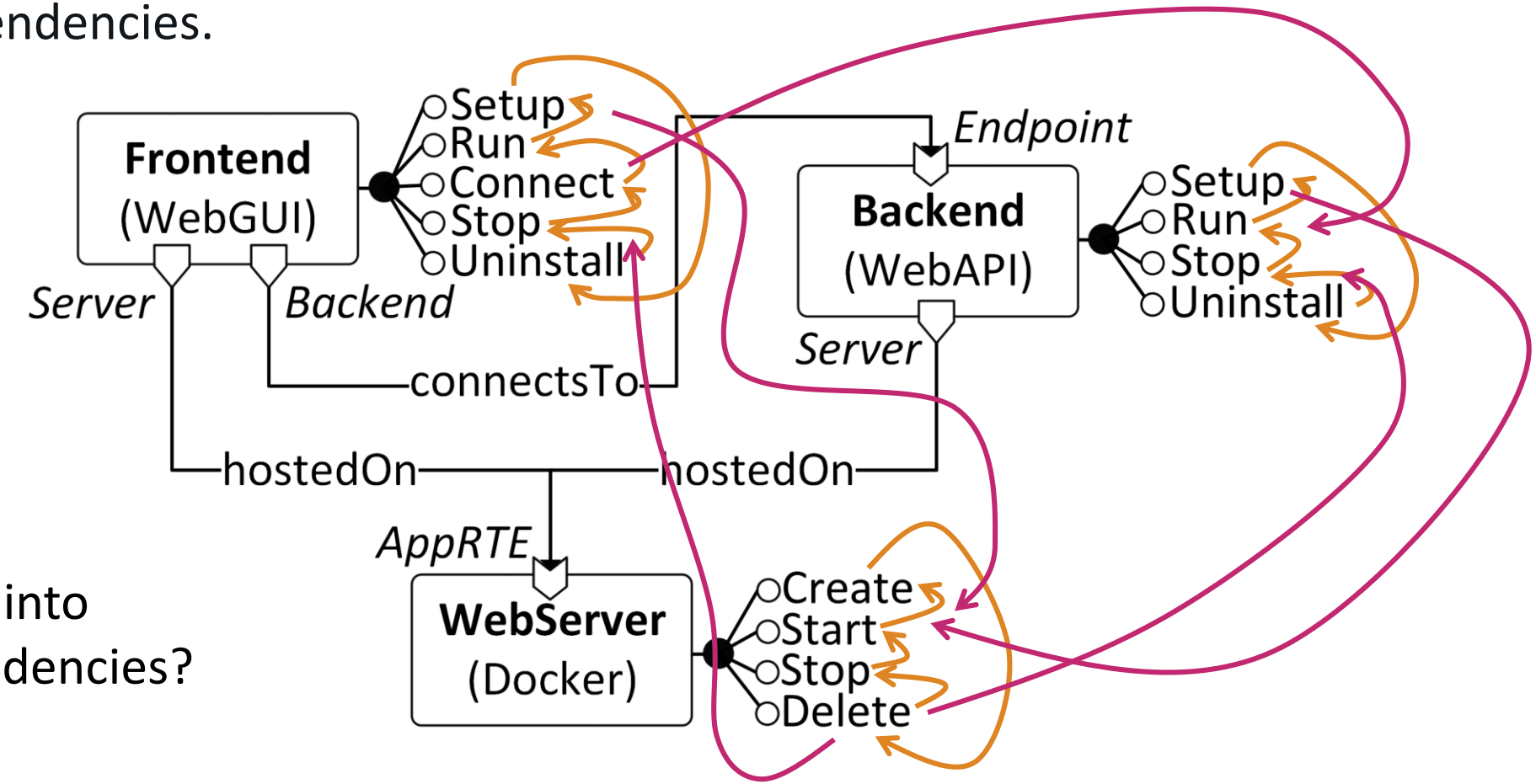
- The OASIS TOSCA standard
- Modelling and analysing cloud application management
- Fault-aware application management protocols
- Conclusions

# Motivations

**Analyse/automate** the **management** of composite cloud applications.

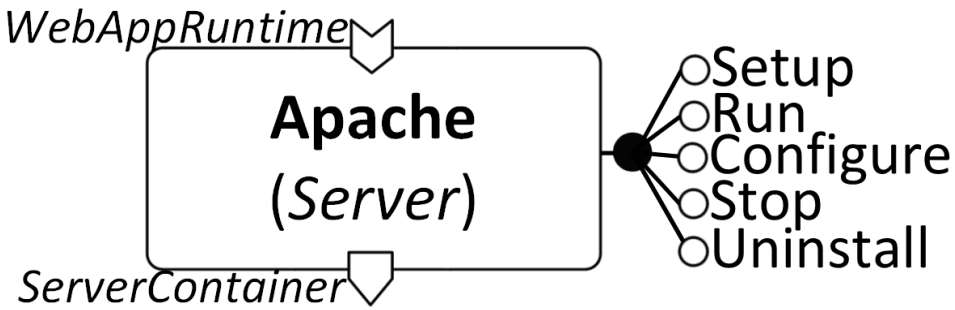
» **Intra-component** dependencies.

» **Inter-component** dependencies.




How to **easily** take into account **all** dependencies?

# Management protocols

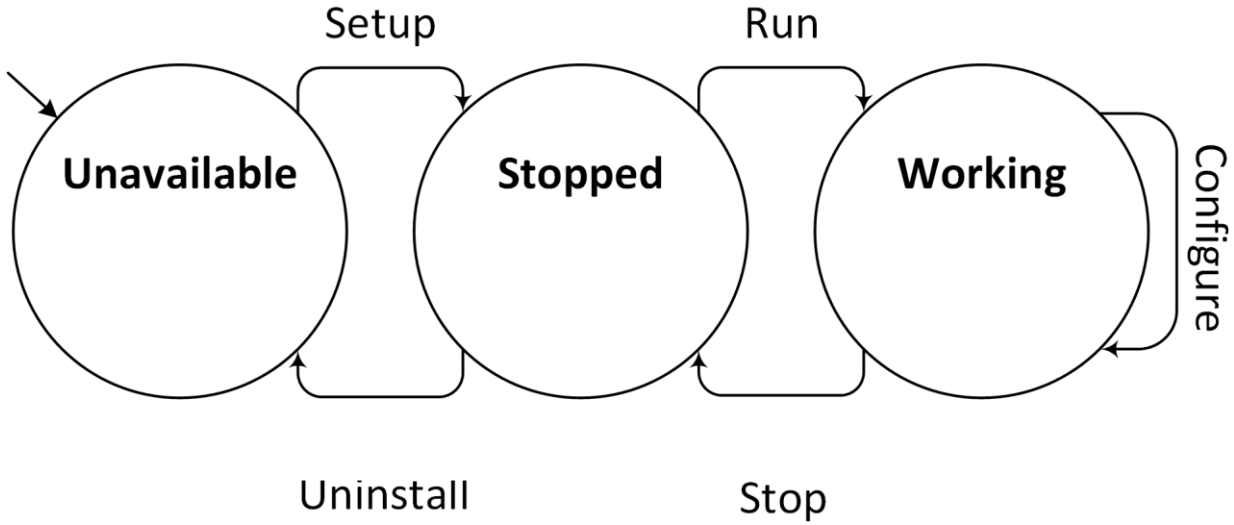



The **management protocol** of a component is a FSM[1].

» Transitions model **intra-component** dependencies.

» Conditions on requirements/capabilities capture **inter-component** dependencies:

- **reqs needed** and **caps offered** in a **state**.
- **reqs needed** to execute a **transition**, and **caps preserved** during its execution.



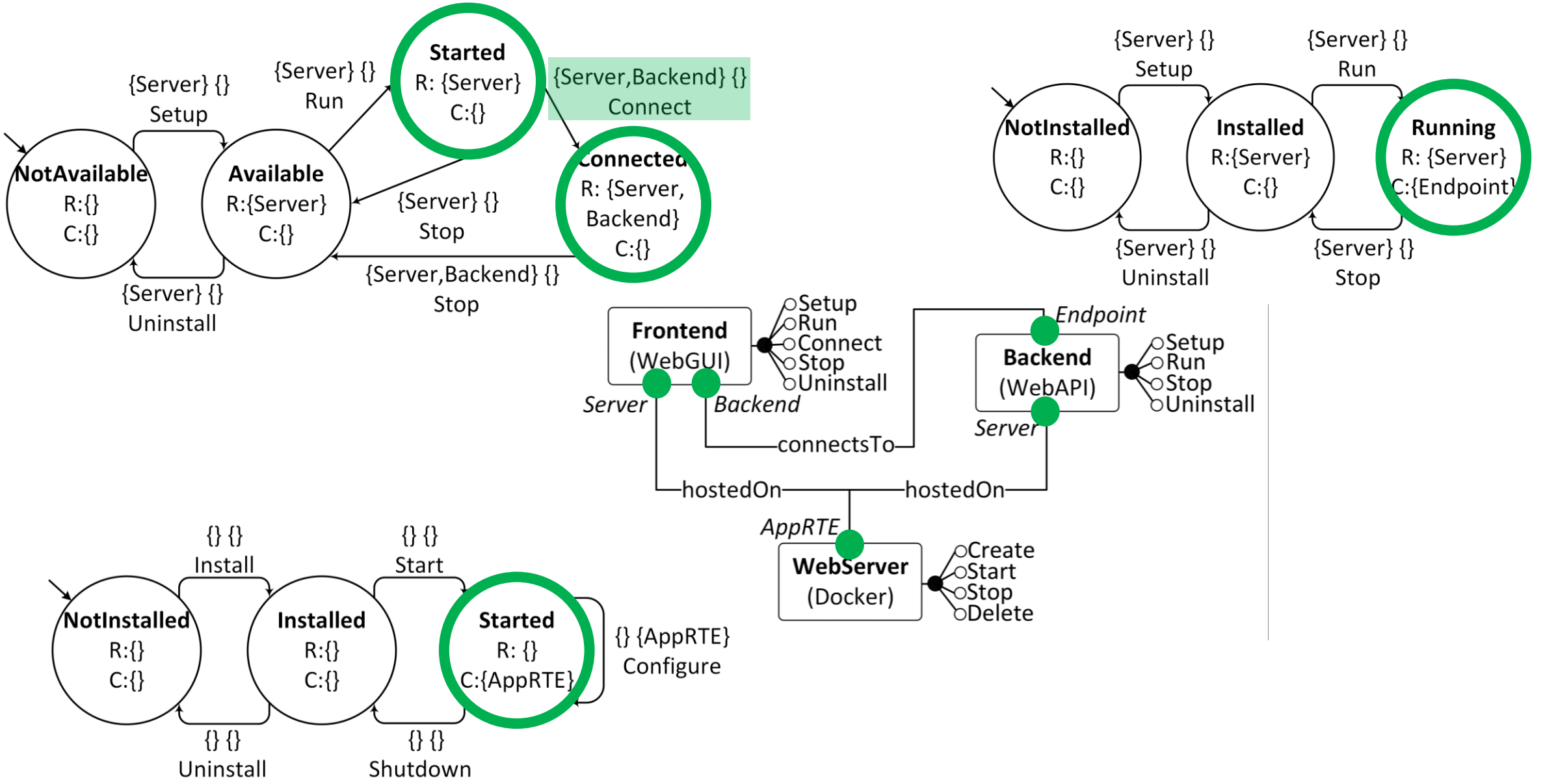

[1] Notions of **well-formedness** and **determinism** of management protocols are defined and can be automatically checked.

# Reasoning with composite applications

The **management behaviour** of a composite application is derived by composing the management protocols of its components.

- A **global state** $G$ is a set containing the current state of each component.
- A global state $G$ is **consistent** iff all the requirements assumed in $G$ are satisfied.
- An **operation can be executed** in $G$ iff all the requirements it needs are satisfied in $G$.

# Example – Consistent global state & operation execution



# Analysing the management of applications

**Validity** of plans

- A **sequence** of management operations $o_1 o_2 \ldots on$ is **valid** from a global state $G_0$ iff $G_0 \xrightarrow{o_1} G_1 \xrightarrow{o_2} G_2 \xrightarrow{o_3} \ldots \xrightarrow{o_n} G_n$ and each $G_i$ is consistent.
- A **workflow plan** is **valid** from a global state $G_0$ iff all its sequential traces are valid from $G_0$.

**Effects** of (valid) plans

- The **effects** of a plan (on states, requirements, capabilities) can be directly determined from global states.
- A valid **plan** is also **deterministic** if all its sequential traces reach the same global state.

**Finding plans** (achieving desired goals)

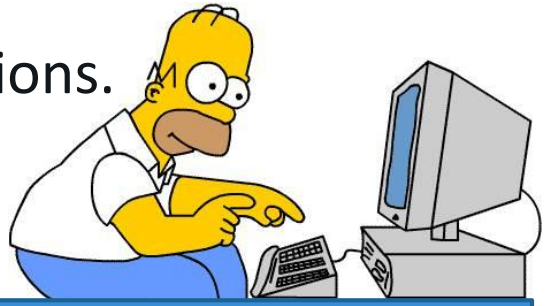
- The problem can be solved with a visit of the graph of reachable global states.

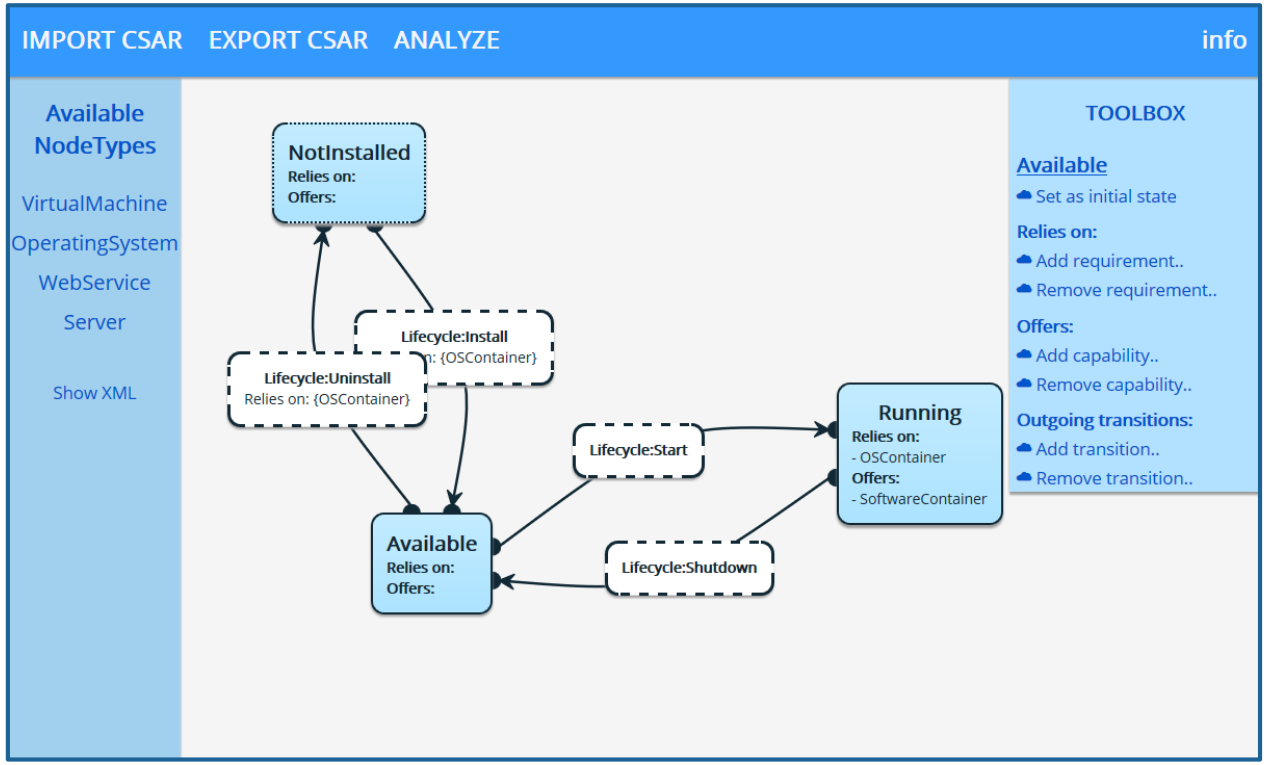
…

# Implementation

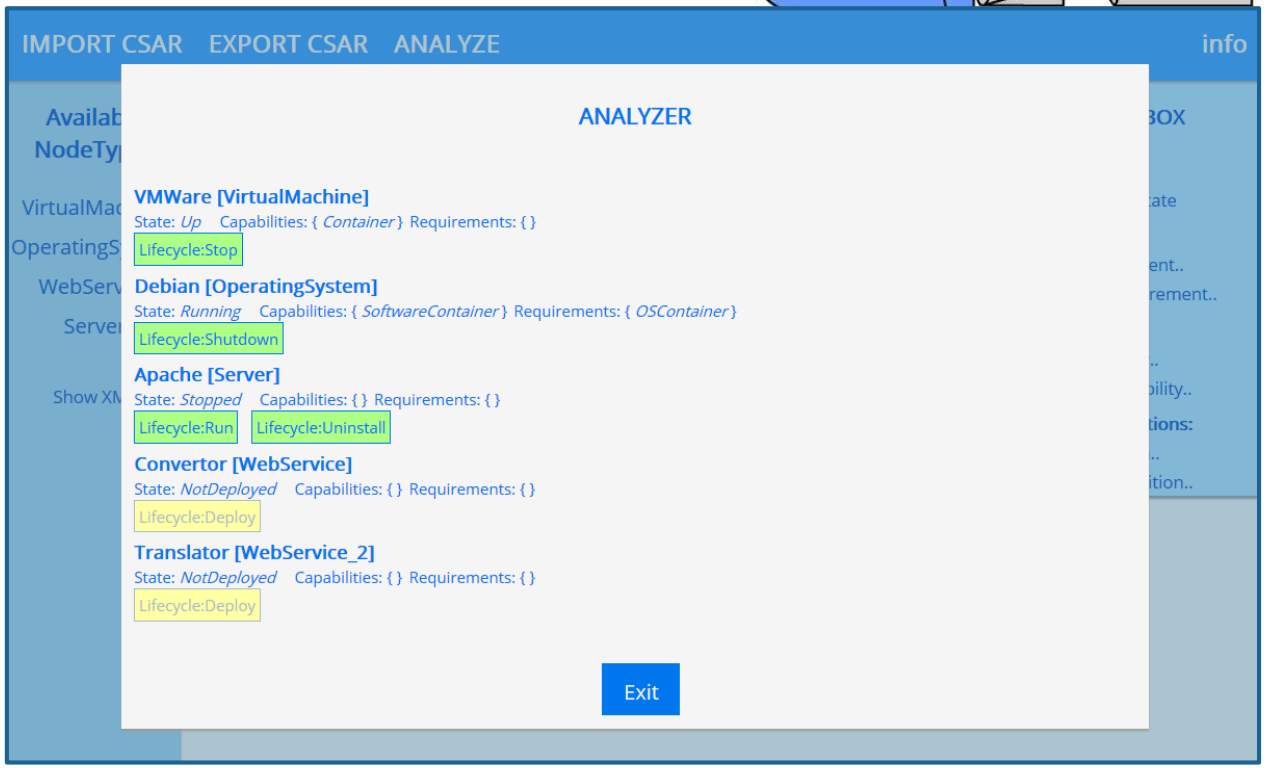
## Barrel[1]

- Web-based **editor/analyser** of management protocols in TOSCA applications.
- **Open-source** and compatible with the OpenTOSCA ecosystem.






edit




analyse

[1] http://ranma42.github.io/MProt

# Case study

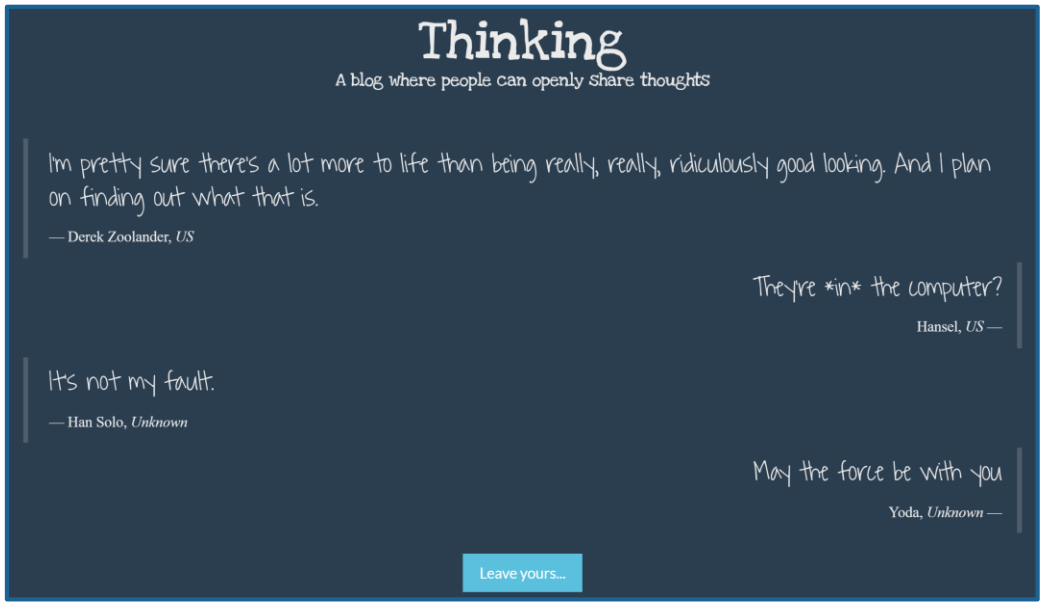
## Thinking

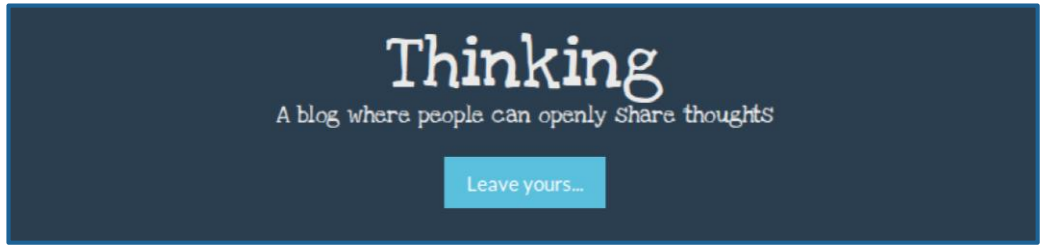
» Real application, made by three components
- **GUI** (deployed on a **NodeJS** Docker cont.)
- **REST API** (deployed on a **Maven** Docker cont.)
- **Mongo** database (running as a Docker cont.)

» **Validation** and **test** of existing deployment plans
- Valid plans effectively deploy application.
- Non-valid plans resulted in crashes/exceptions.

» **Planning**
- Valid plan to undeploy GUI and REST API (only)
- Effectively resulted in undeploying them.

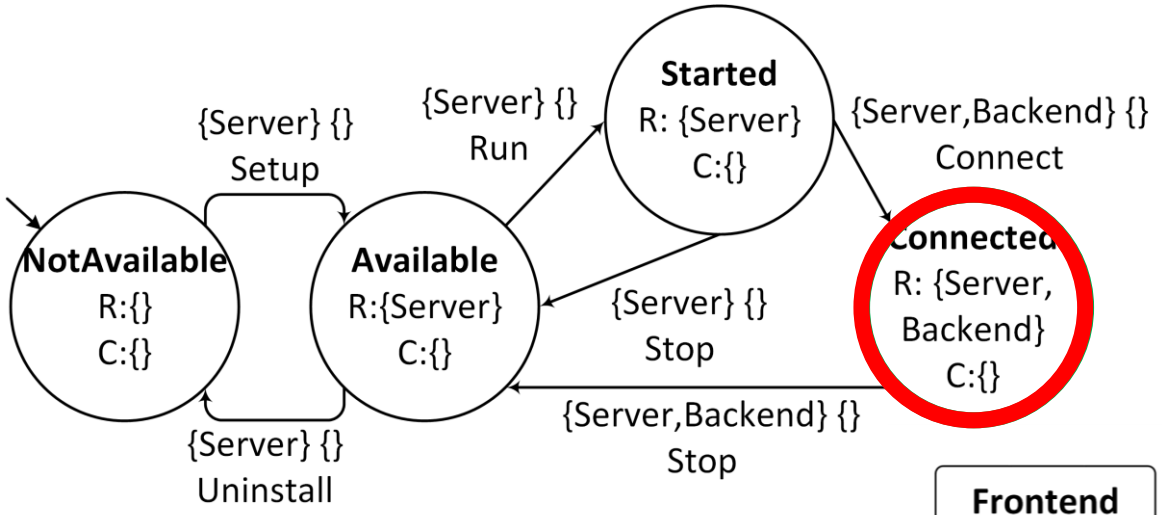
```
jacopo@yellow:~$ docker ps -a
CONTAINER ID   IMAGE  COMMAND                  CREATED         STATUS         PORTS
f385a566c619   mongo  "/entrypoint.sh mongo"  17 minutes ago Up 17 minutes 27017/tcp
jacopo@yellow:~$
```

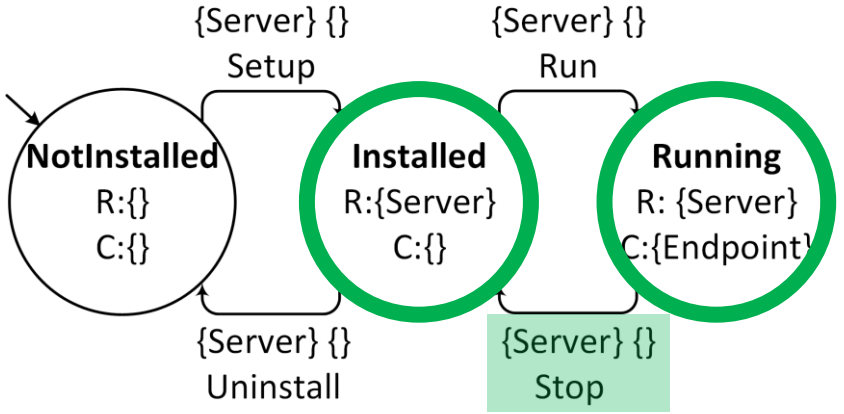


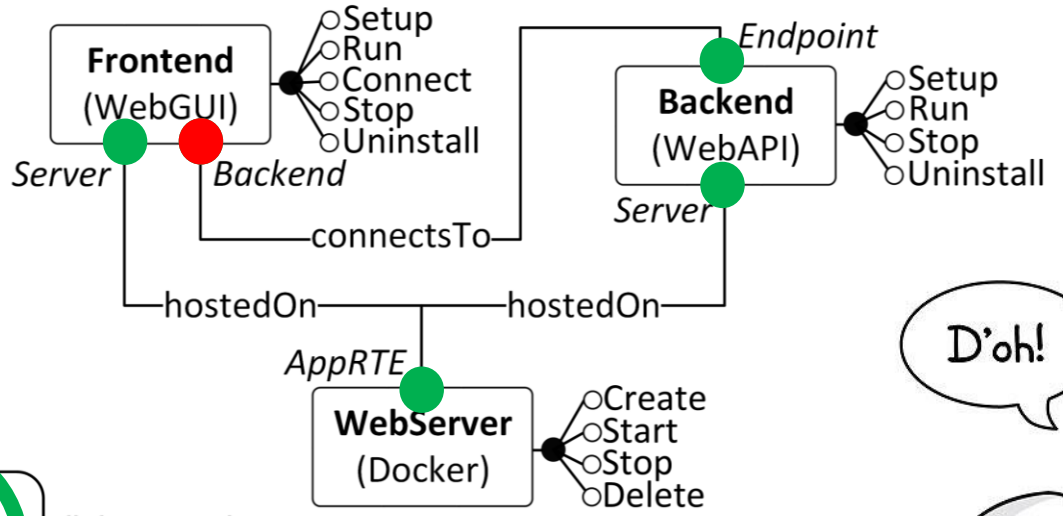



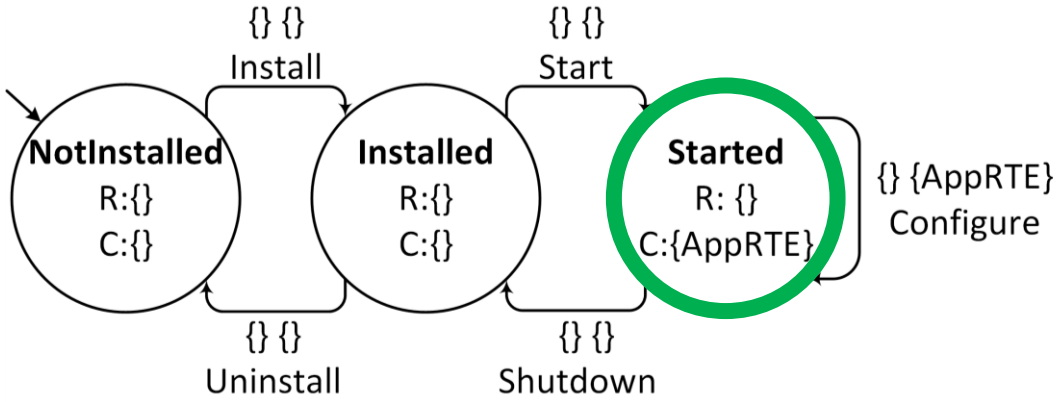

```
INFO  [2016-08-04 14:38:05,071] org.mongodb.driver.cluster: Exception in monitor
 thread while connecting to server unknown:27017
! java.net.UnknownHostException: unknown: unknown error
! at java.net.Inet6AddressImpl.lookupAllHostAddr(Native Method)
! at java.net.InetAddress$2.lookupAllHostAddr(InetAddress.java:928)
! at java.net.InetAddress.getAddressesFromNameService(InetAddress.java:1323)
! at java.net.InetAddress.getAllByName0(InetAddress.java:1276)
! at java.net.InetAddress.getAllByName(InetAddress.java:1192)
! at java.net.InetAddress.getAllByName(InetAddress.java:1126)
! at java.net.InetAddress.getByName(InetAddress.java:1076)
! at com.mongodb.ServerAddress.getSocketAddress(ServerAddress.java:186)
! ... 5 common frames omitted
```

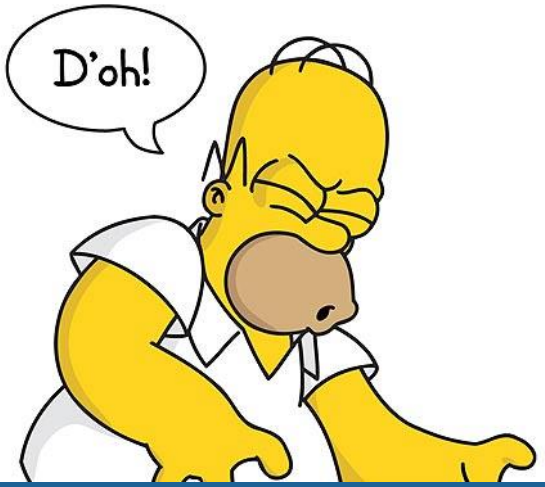
# Another example – (In)Consistent global state



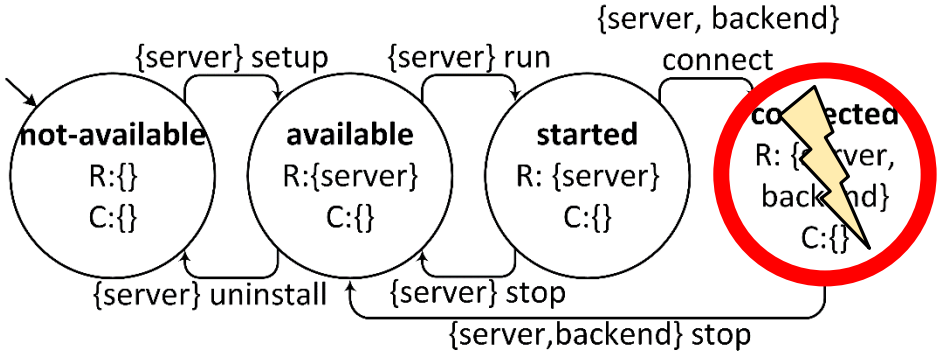



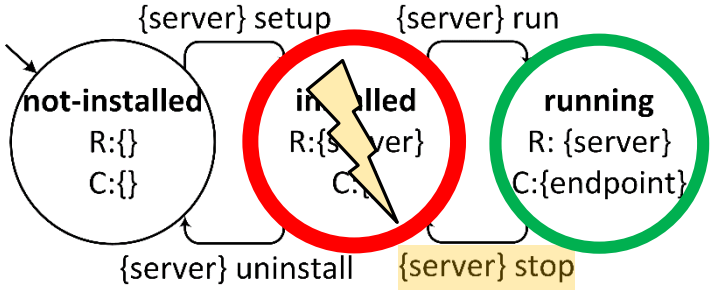
# Outline

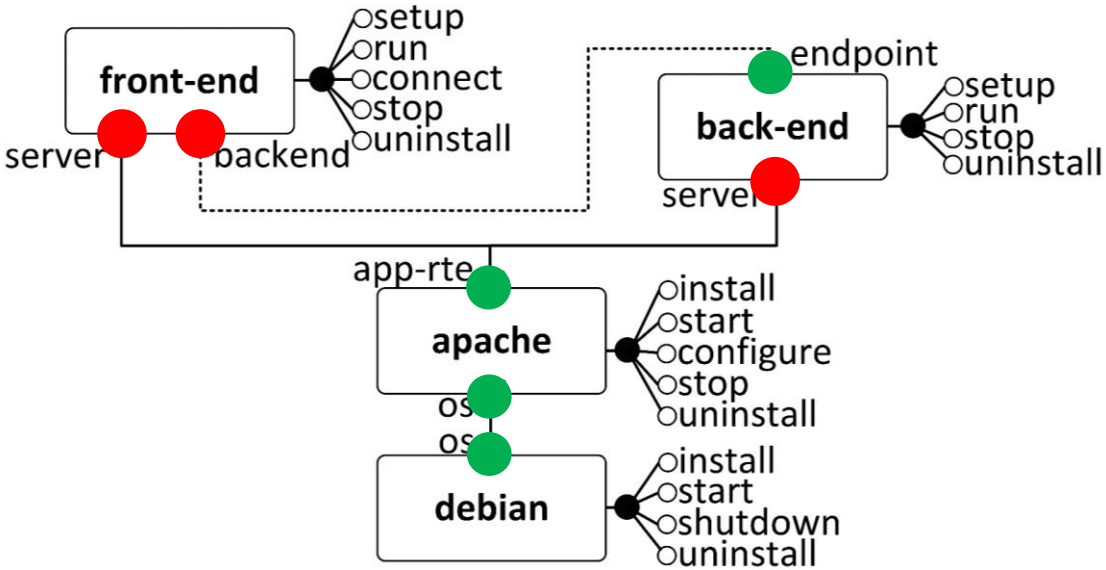
- The OASIS TOSCA standard
- Modelling and analysing cloud application management
- Fault-aware application management protocols
- Conclusions

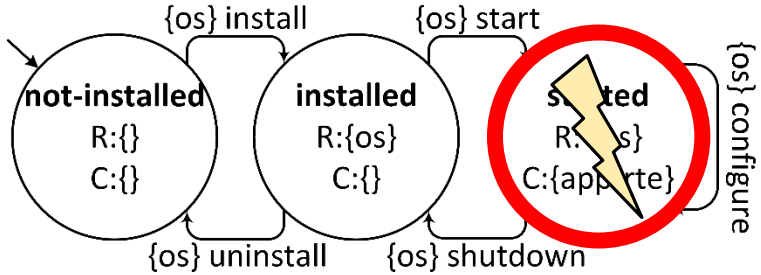
# Motivations

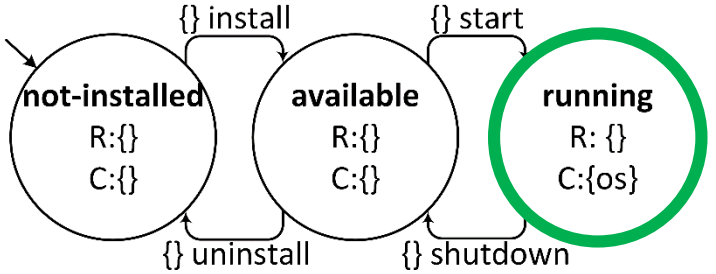
How to handle the **fault of requirements**?




Effects of **misbehaving components**?

# Our approach

**Fault-aware management protocols** permit

- modelling how nodes behave when faults occurs, and
- analysing/automating application management in presence of faults.

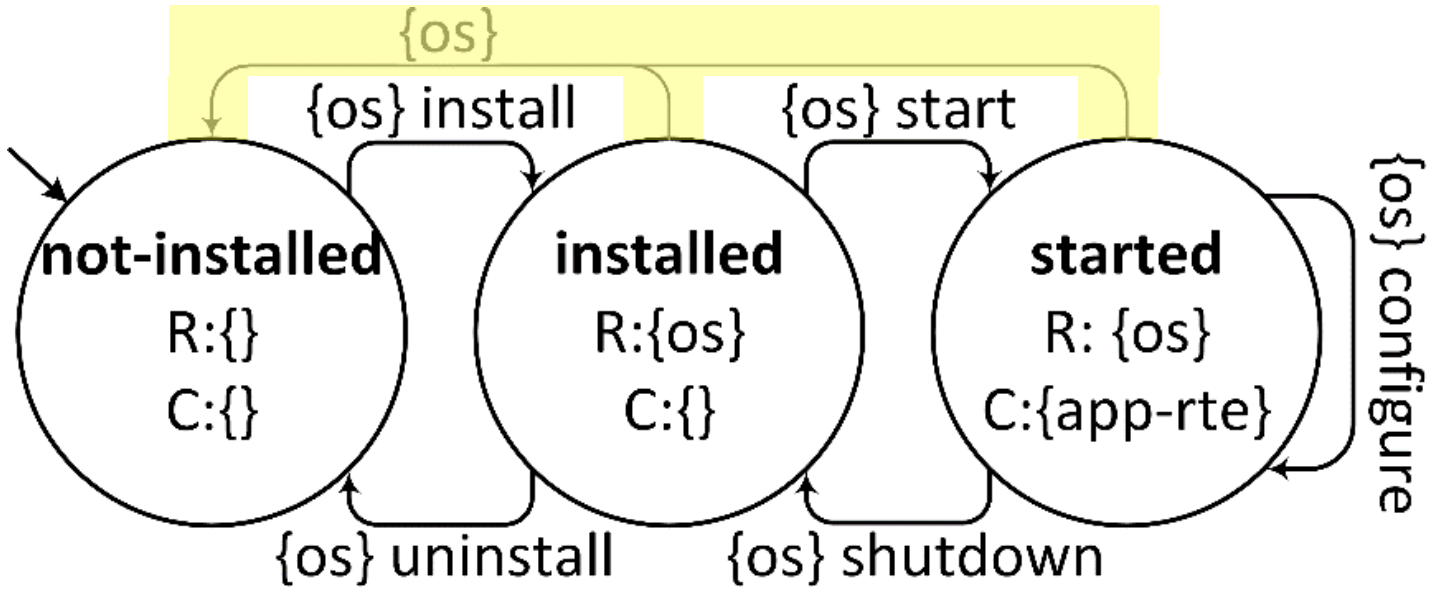
**Unexpected behaviour**

- naturally modelled in (fault-aware) management protocols
- to permit analysing the (worst possible) effects of a misbehaving component.

**Planning** how to **hard recover** applications that are stuck

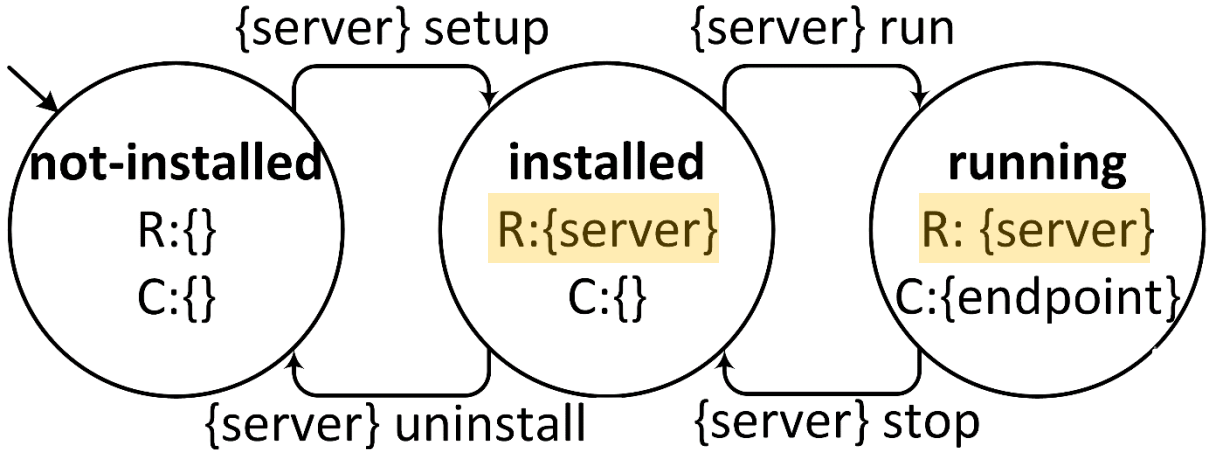
- since a fault was not properly handled, or
- because of a misbehaving component.

# Fault-aware management protocols



# Default handling

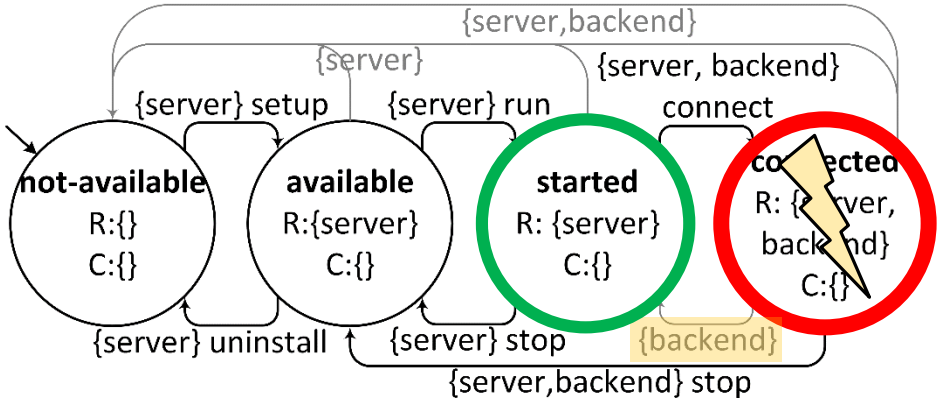
Application designers may leave the handling of some faults unspecified.

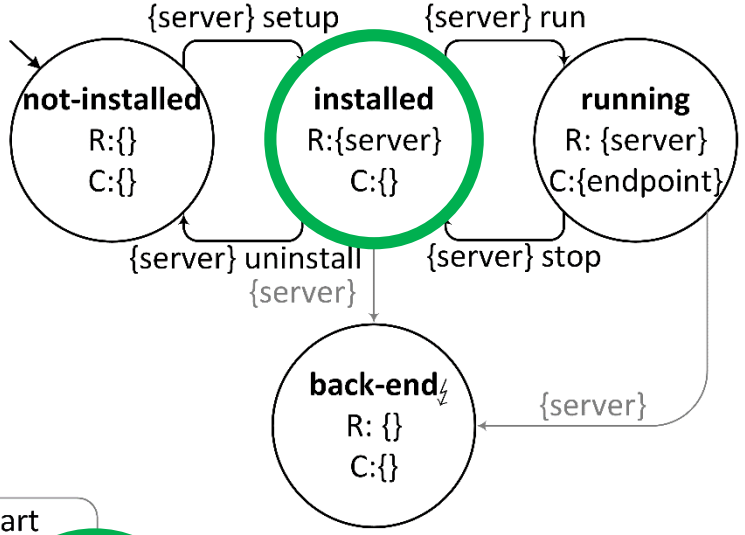


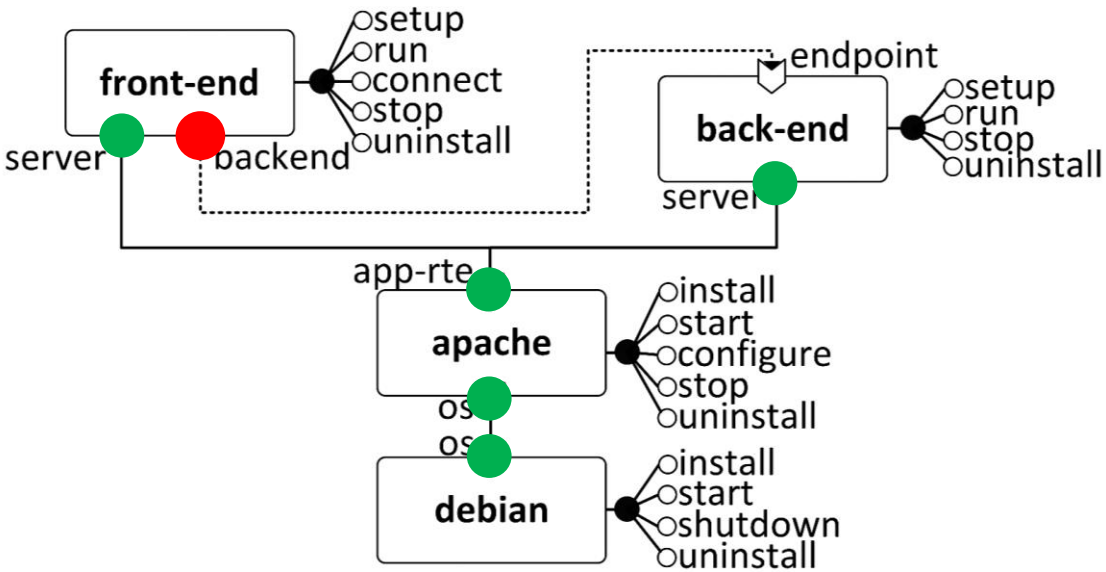

**Default handling** to a **sink state** that requires/provides nothing (worst-case assumption).

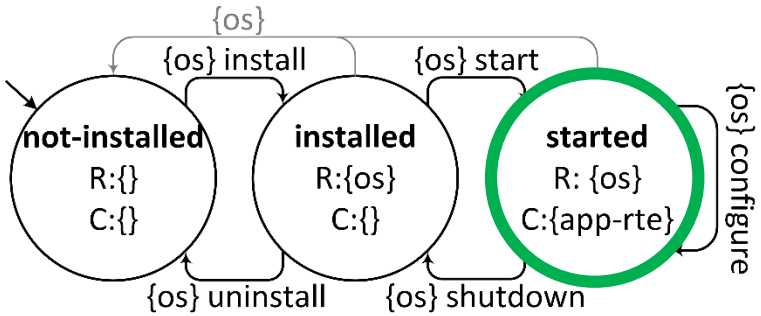
# Reasoning with composite applications (and with faults)

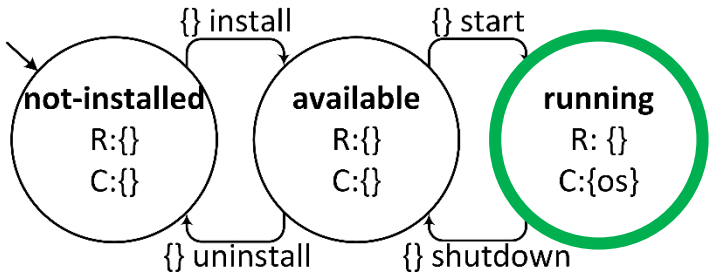
How to handle the **fault of requirements**?



# Analysing the management of applications

**Validity** of plans

» …

**Effects** of (valid) plans

» …

**Finding plans** (achieving desired goals)

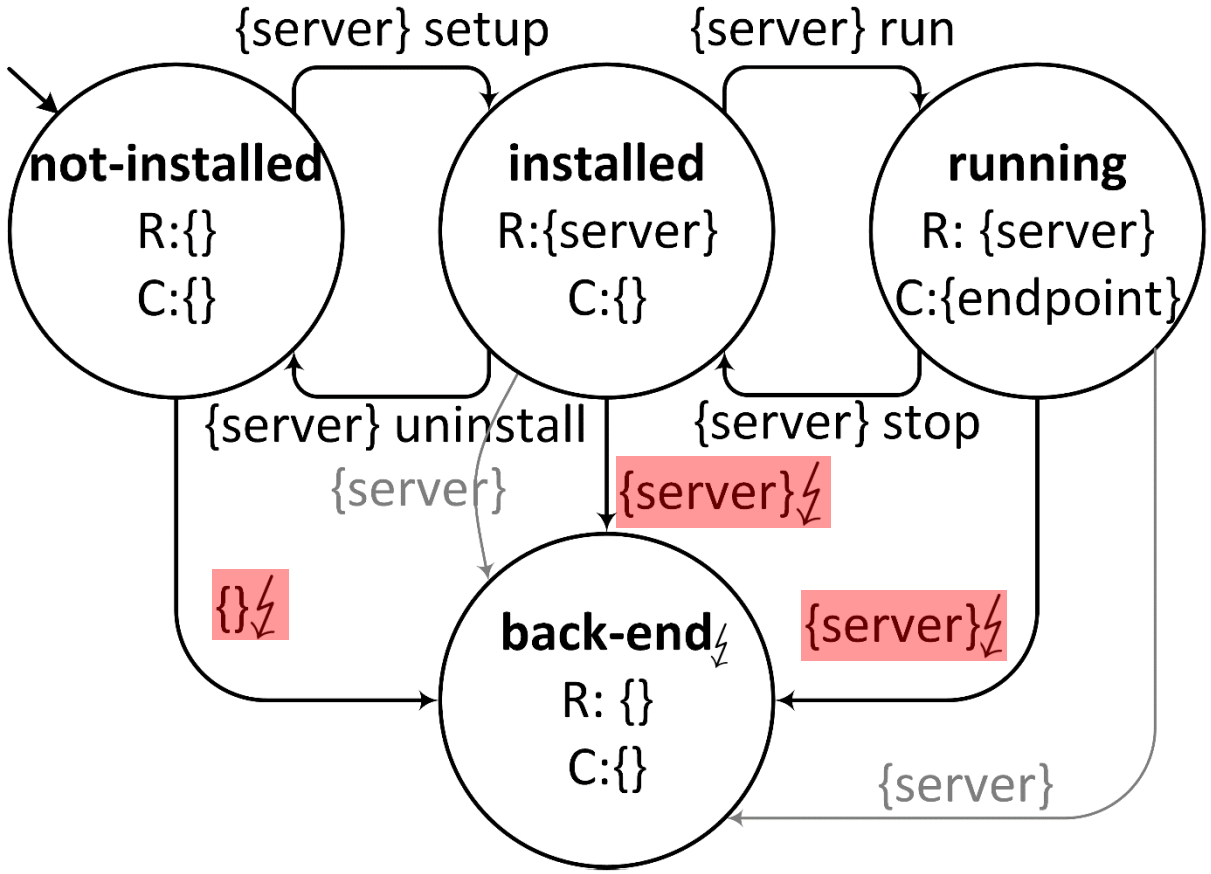
» …

…

All previously introduced analyses can still be **automatically performed**
(now also taking into account **faults**)

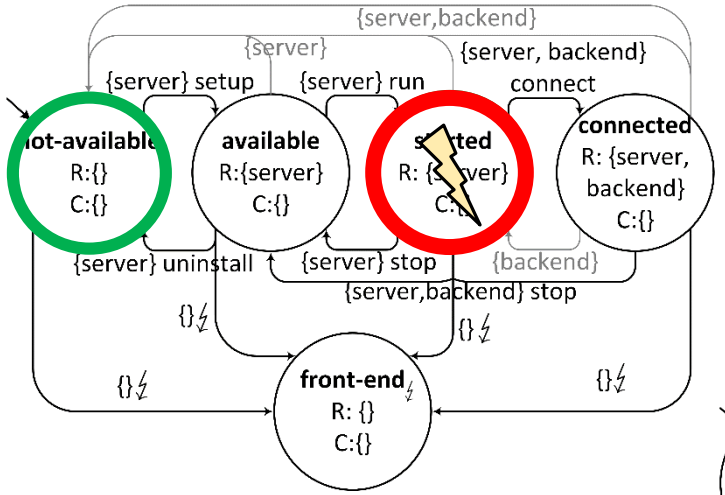
# Dealing with «misbehaving components»

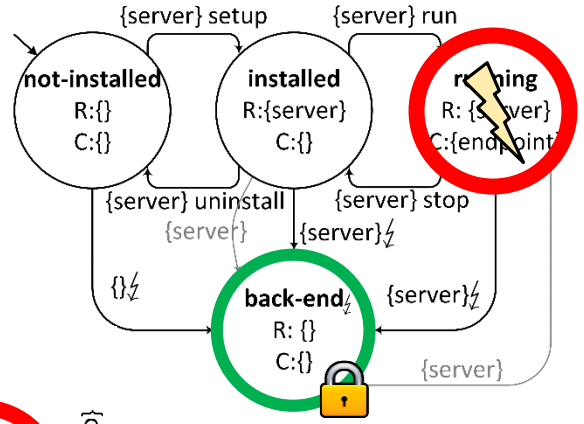
The **unexpected behaviour** of a component can be modelled with a special «**crash**» operation..

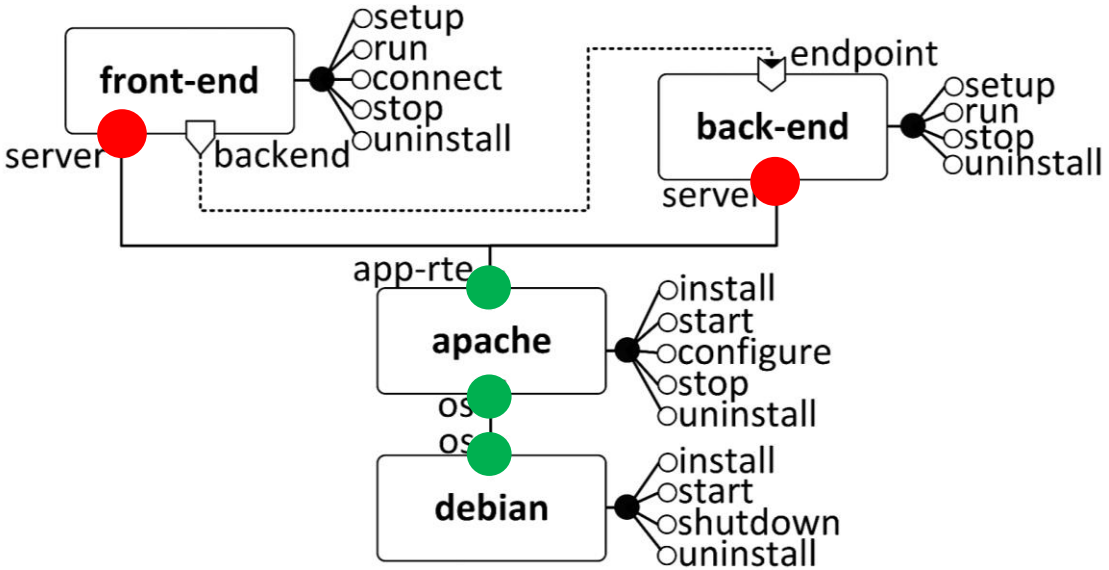


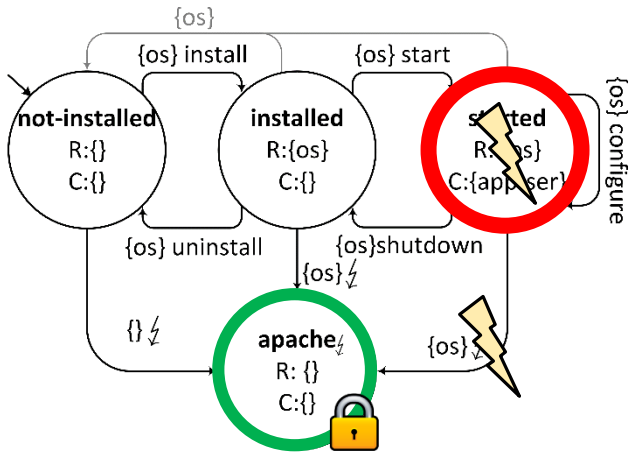

..leading to a **sink state** that provides/requires nothing (worst-case assumption).

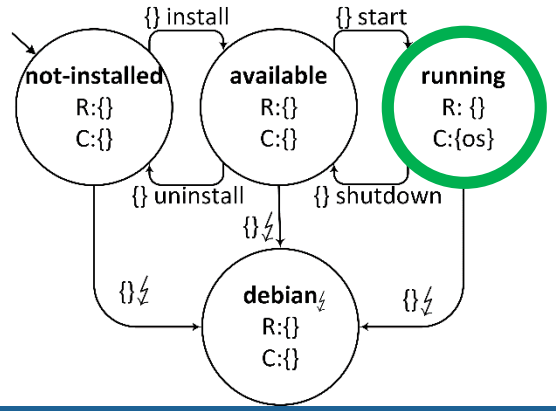
# Dealing with misbehaving components



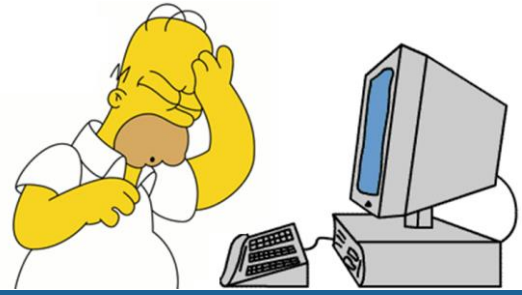




Effects of **misbehaving components**?



# Hard recovery

**Recovery plans** can be generated automatically.

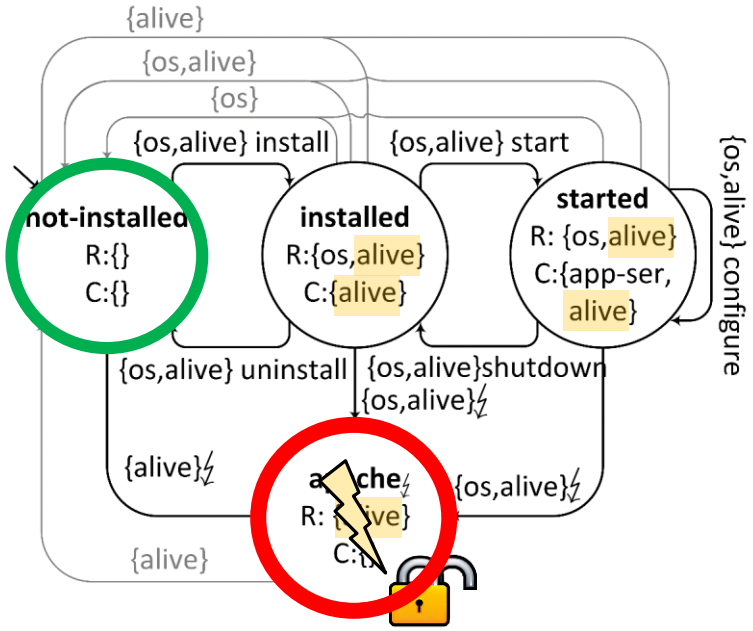
**Idea** (from our experience):

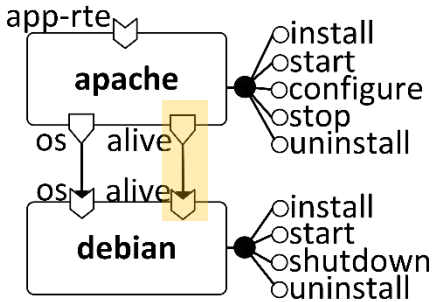
Machine **stuck**, not responding → Forcibly **restart** it — *by **resetting** the whole **system***

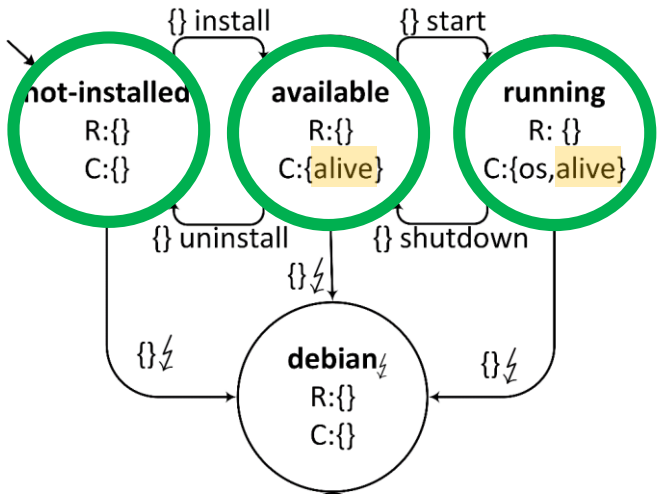
↓ ↓

Node **stuck**, not responding → Forcibly **restart** it — *by **resetting** the **container** node, hence resetting all nodes it contains*

# Example - Hard recovery



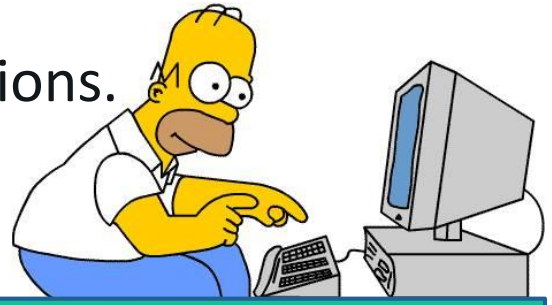



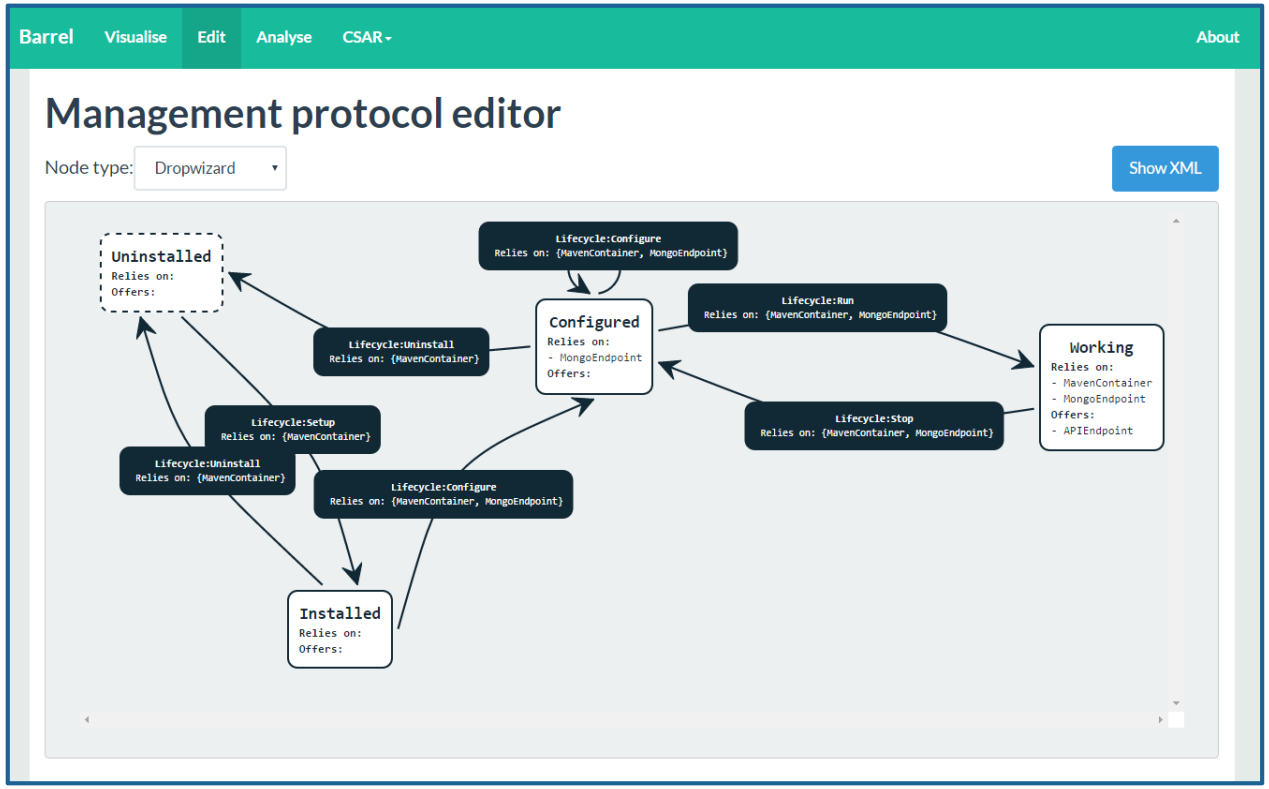



# Implementation

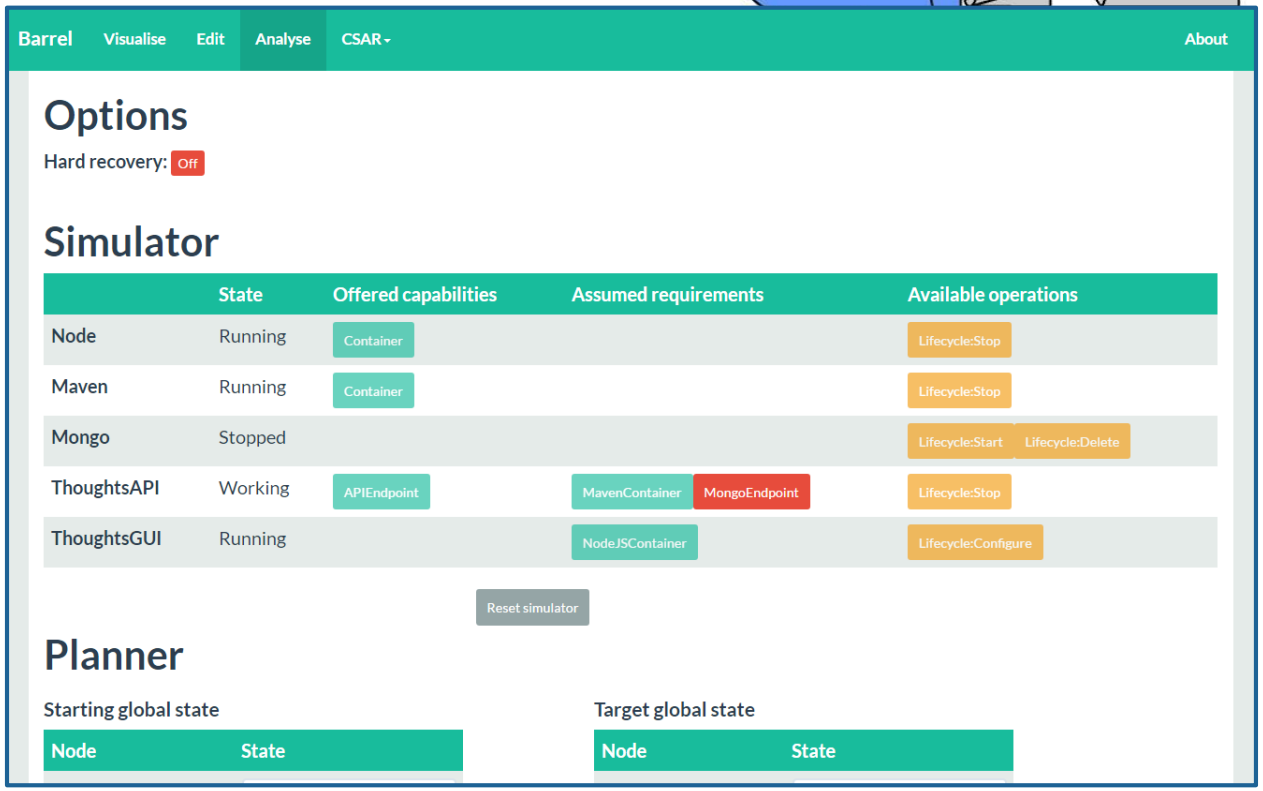
## Barrel[1]

- Web-based **editor/analyser** of management protocols in TOSCA applications.
- **Open-source** and compatible with the OpenTOSCA ecosystem.






edit




analyse

[1] http://di-unipi-socc.github.io/barrel

# Case study

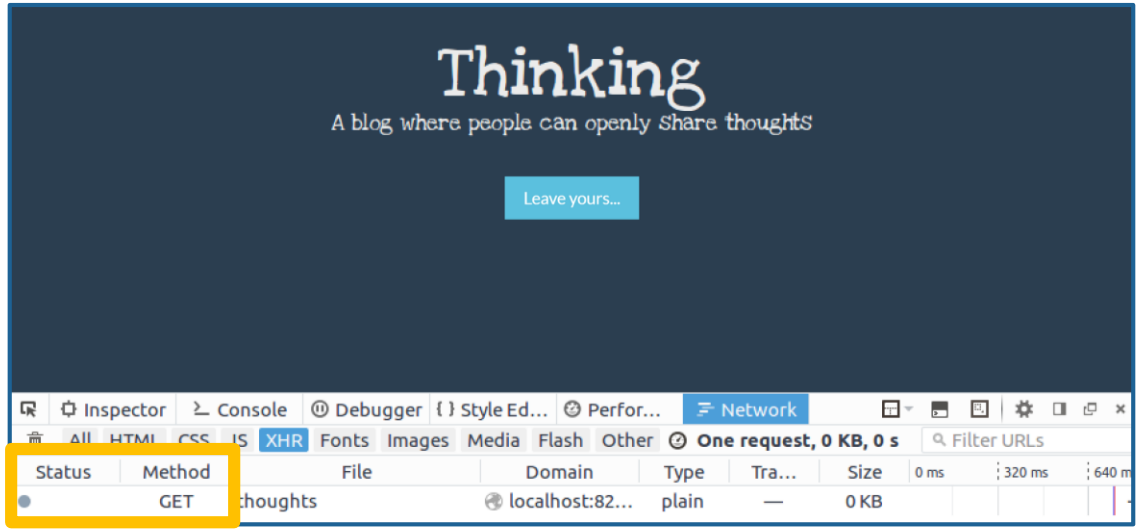
## Thinking

» Real application, made by three components
- **GUI** (deployed on a **NodeJS** Docker cont.)
- **REST API** (deployed on a **Maven** Docker cont.)
- **Mongo** database (running as a Docker cont.)

» **Validation** and **test** of existing deployment plans

» Effects of **misbehaving components**
- e.g., crashed API.

» **Planning**
- e.g., hard recovery of crashed API.




REST API does not return any answer when invoked

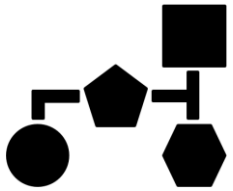
# Implementation (2)

**DEMO**

# Outline

- The OASIS TOSCA standard
- Modelling and analysing cloud application management
- Fault-aware application management protocols
- Conclusions

# Conclusions



**Modelling**
composite cloud applications.

**Management protocols**, which are a **modular**, **compositional**, and **fault-aware** modelling for the management behaviour of application components.

**Analysing**
composite cloud applications.

Techniques for **analysing** and **automating** the management of composite applications (e.g., validity of plans, effects of plans, planning, hard recovery, etc.).

**independent from the employed topology model**

# Future work

**Faults generated during transitions?**


**Cost?**

**QoS?**

# Thank you!

# References


## TOSCA

A. Brogi, J. Soldani, P. Wang. **TOSCA in a nutshell: Promises and perspectives**. 3rd European Conference on Service-Oriented and Cloud Computing (ESOCC 2014), 2014. [link]

## Management protocols

A. Brogi, A. Canciani, J. Soldani. **Modelling and analysing cloud application management**. 4th European Conference on Service-Oriented and Cloud Computing (ESOCC 2015), 2015. [link]

A. Brogi, A. Canciani, J. Soldani, P. Wang. **A Petri net-based approach to model and analyze the management of cloud applications**. Transactions on Petri Nets and other models of Concurrency, volume 11, pages 28-48, 2016. [link]

## Fault-aware management protocols

A. Brogi, A. Canciani, J. Soldani. **Fault-aware application management protocols**. 5th European Conference on Service-Oriented and Cloud Computing (ESOCC 2016), 2016. [link]