# Autonomic reconfiguration - decision making

## Advanced distributed systems


Hélène Coullon


Associate professor at IMT Atlantique, France


Inria researcher, France


Adjunct professor at UiT, Tromsø, Norway


April 19th, 2021

# Overview

1. Reminder of previous talks
2. How to further automate?
3. Decision making
4. State of the art
5. Automatic generation of CONCERTO programs
6. Conclusion

# Table of Contents

# Reminder

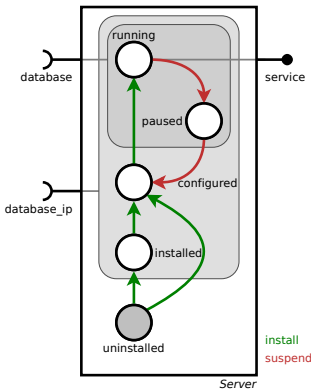
- Tools to express and answer [WHAT][WHERE][HOW][WHEN] in DevOps problems
  - deployment
  - management / reconfiguration
- Enriched with [LIFECYCLE][DEPENDENCIES] modeling
  - efficiency
  - separation of concerns

## MADEUS and CONCERTO

- CONCERTO reconfiguration model and its implementation,
- MADEUS is a subpart of CONCERTO,
- parallelism expressiveness,
- formal semantics and automatic coordination.

# Control components

Written by the component developers

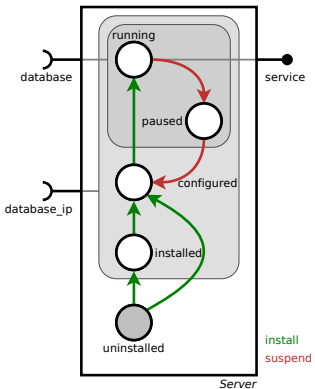



## Internal net **[LIFECYCLE]**

- places = milestones
- transitions = actions to perform
  - concretely: scripts are attached to transitions
  - in the model: exact nature/effects of actions not represented, only coordination

# Control components

Written by the component developers




## Interfaces [DEPENDENCIES]

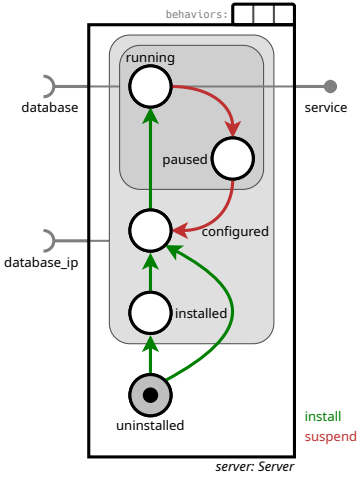
- data or service ports
  - use ports = requirements
  - provide ports = provisions
  - during execution: active/inactive
- behaviors
  - subset of transitions
  - during execution: active/inactive

# Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

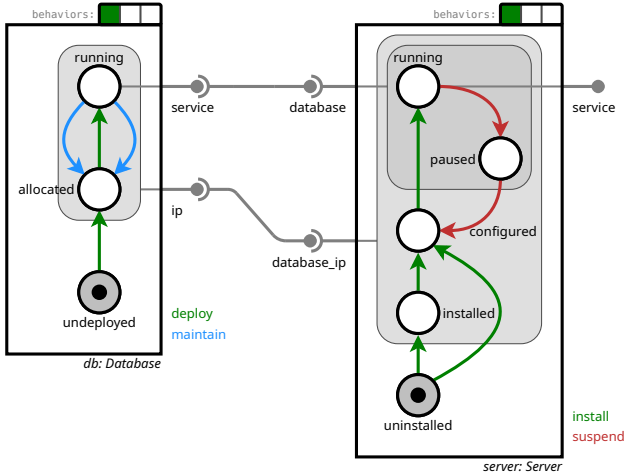


# Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

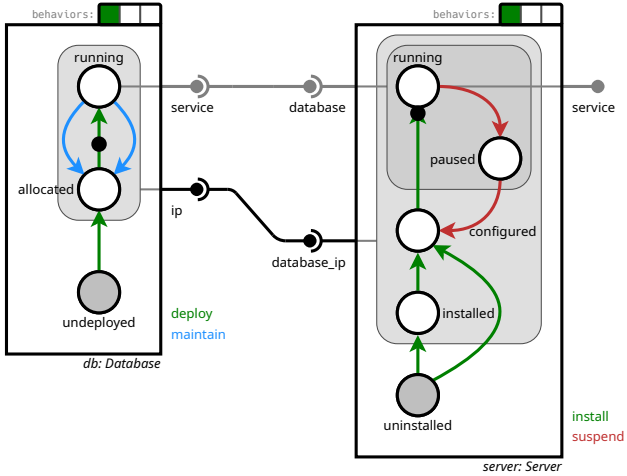


# Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

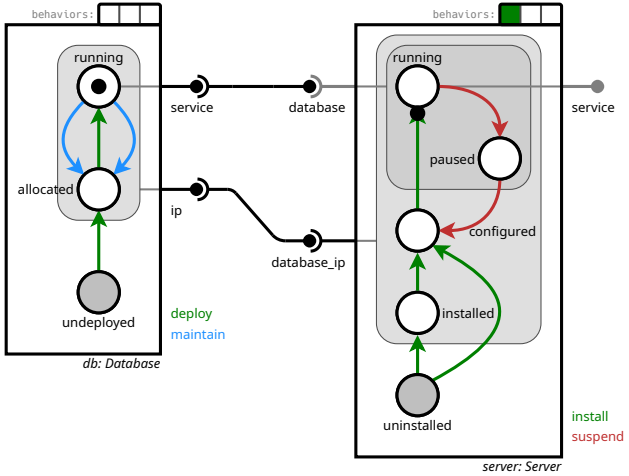


# Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

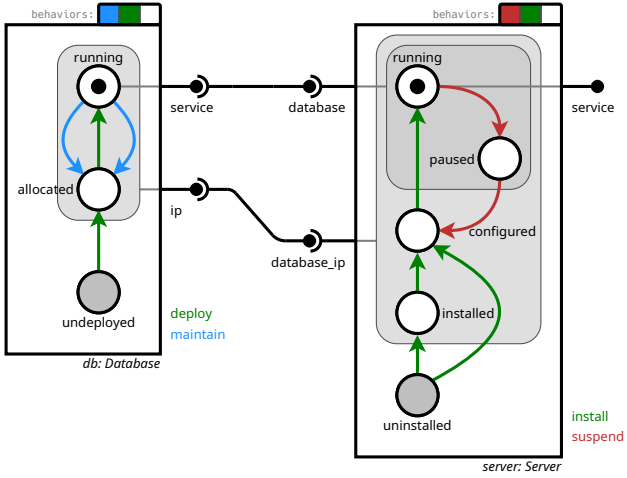


# Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

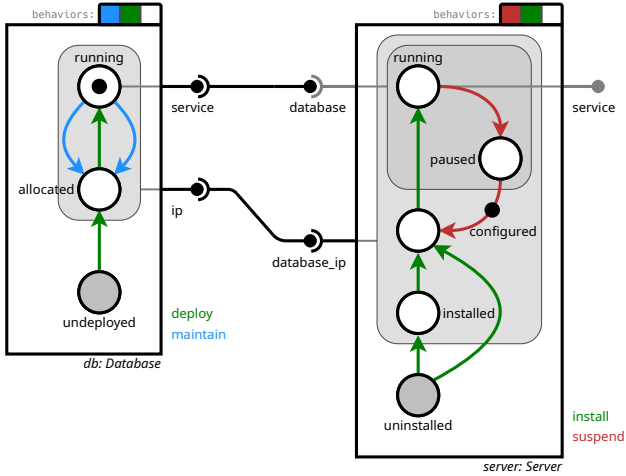


# Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

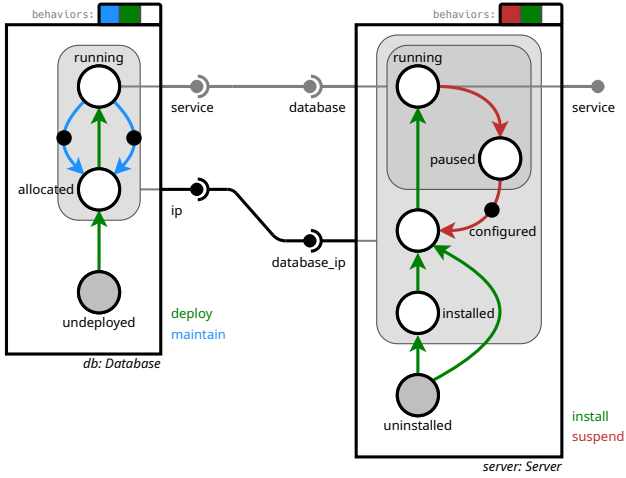


# Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```



# Table of Contents

# How to further automate?

## What have been automated so far?

- the coordination between the lifecycles,
- the coordination between actions of a given lifecycle,
- the set of reconfiguration instructions (add, remove etc.).

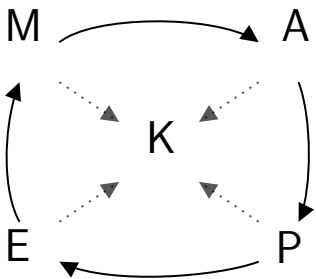
## What is manually written?

- the lifecycle of each component and their dependencies, component developer
- the new state to reach, reconfiguration developer
  - involved components,
  - their final connections,
  - their desired final state,
- the reconfiguration program/plan to reach this new state. reconfiguration developer

State1 $\longrightarrow$ State2 $\longrightarrow$ State3 ...

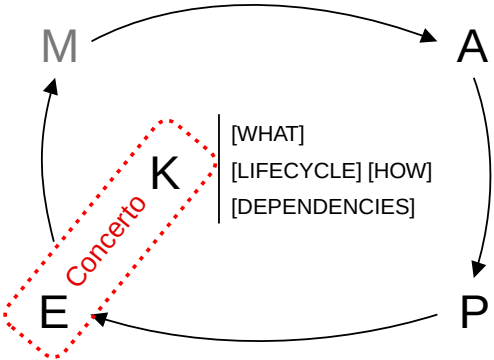
# The autonomic loop MAPE-K

- **(M)onitoring** of the system and its environment,
- **(A)nalysis** of the current situation and decides a target state,
- **(P)lanning** of the set of modifications and operations to reach the new state,
- **(E)xecution** of the plan,
- the **(K)nowledge** is shared by all steps, it contains models and data.



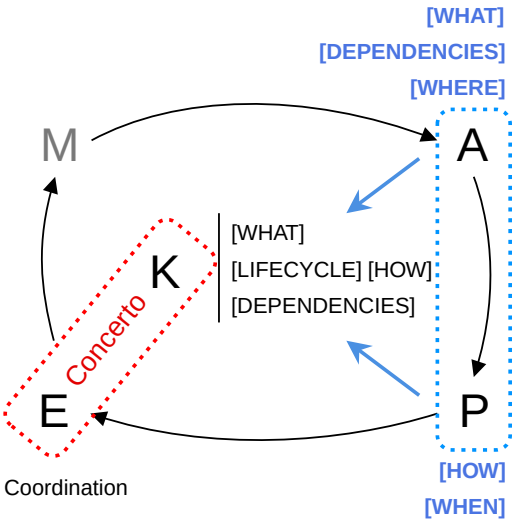

*The vision of autonomic computing. J. O. Kephart and D. M. Chess. In Computer, 2003.*

# MAPE-K



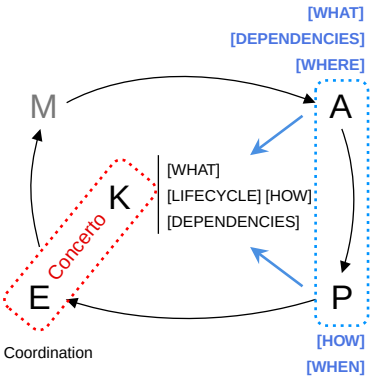

Coordination

# MAPE-K


[WHAT]
[DEPENDENCIES]
[WHERE]
M
A
K
[WHAT]
[LIFECYCLE] [HOW]
[DEPENDENCIES]
Concerto
E
P
[HOW]
[WHEN]
Coordination

# Examples of decision problems

- configuration problem **[WHAT][DEPENDENCIES]**
- placement, bin-packing problem **[WHERE]**
- scheduling problem **[WHEN][HOW]**



# Table of Contents

# Satisfaction or optimization problems?

## Dissociate two categories of decision problems

- satisfaction problem
  - assign values to variables while satisfying a set of constraints
- optimization problem
  - add one or multiple objective functions to the satisfaction problem

# Example satisfaction/optimization problem

**Satisfaction problem**

- $D$ a set of comuting devices with a capacity $C$ (e.g., CPU),
- $S$ the set of services to be hosted on devices,
- $s(i), i \in S$ the size of the service,
- assign all services to devices by assigning $x_{i,j} = 1$ if the service $i$ is hosted on device $j$,
- with the following constraints

$$(1) \quad \sum_{i \in S} s(i) x_{i,j} \leq C_j, \forall j \in D \qquad \text{(do not exceed the capacity of devices)}$$

$$(2) \quad \sum_{j \in D} x_{i,j} = 1, \forall i \in S, x_{i,j} \in \{0,1\} \qquad \text{(each service is hosted by a single device)}$$

**Optimization problem**

$$(3) \quad \textit{minimize} \sum_{j \in D} y_j \qquad \text{(minimize the number of devices used to host services)}$$

with $y_j = 1$ if the device hosts at least one service

# Examples of solutions and solvers

**Depends on the logic/theories to use for variables and constraints**

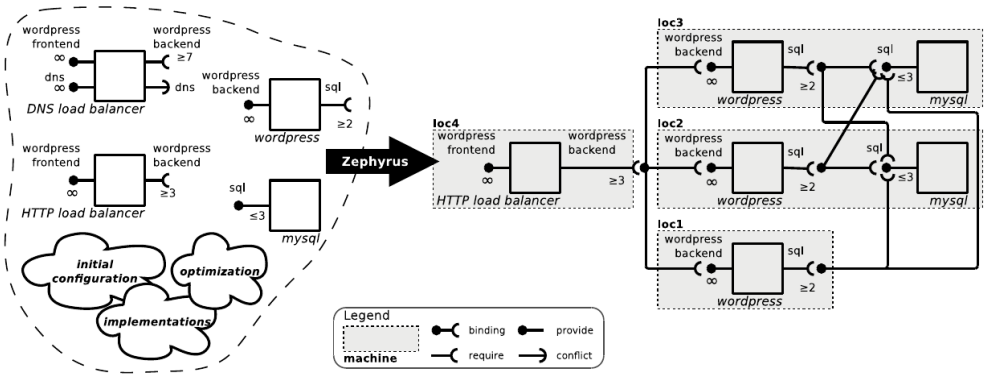
- Satisfaction problems
  - Boolean satisfaction problem (SAT): Glucose, OR-tools etc.
  - Constraint Satisfaction Problem (CSP) / Constraint Porgramming (CP): Choco, GeCode, OR-tools etc.
  - Satisfiability Modulo Theories (SMT): z3, CVC etc.
- Optimization
  - Integer Linear Programming (ILP): OR-tools, PULP, CPLEX etc.
  - Constraint Optimization Problem (COP) / CP: Choco, GeCode, OR-tools etc.
  - Ad-hoc heuristics: greedy, first-fit, meta-heuristics, hyper-heuristics etc.

# Table of Contents

# Many contributions in the literature

- (A)nalysis: placement and configuration problems [WHAT][WHERE]
  - CSP, SAT, SMT: *[zephyrus1][zephyrus2][engage][btrplace][confsolve]*
  - heuristics: *[heuristic1]* and many others!
- (P)lanning: sheduling and planning problems [HOW][WHEN]
  - CSP, SAT, SMT: *[concerto]*
  - ad-hoc search algorithm: *[metis]*
  - heuristics: *[heuristic2]* and many others!

# Zephyrus (Aeolus) - Analysis

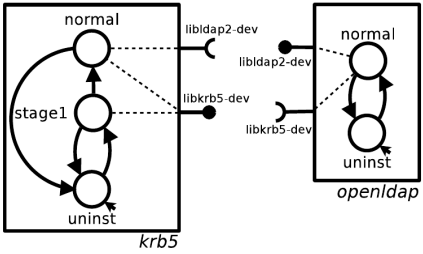



Specific constraint language → MiniZinc constraint modeling language → GeCode solver

## Constraint example

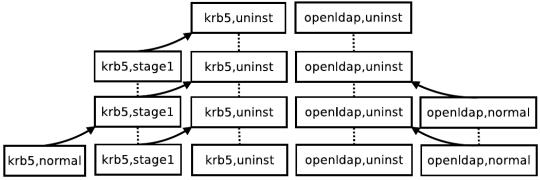
Each MySQL instance shouldn't serve the needs of more than 3 Wordpress instances

# Metis (Aeolus) - Planning

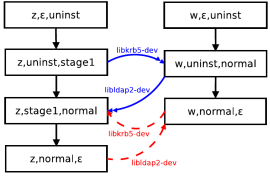



## Plan synthesis

- Inputs:
  - current state = nothing (deployment)
  - target state
  - set of Aeolus components
  - set of possible instructions
- output = reconfiguration plan (program)




Reachability graph

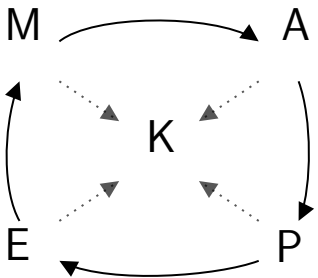



Abstract plan

# Table of Contents

# The autonomic loop MAPE-K

- **(M)onitoring** of the system and its environment,
- **(A)nalysis** of the current situation and decides a target situation,
- **(P)lanning** of the set of modifications and operations to reach the new state,
- **(E)xecution** of the plan handled by CONCERTO,
- the **(K)nowledge** through CONCERTO.




## Planning automation

Comparable to Metis but

- using SMT solver instead of ad-hoc algorithms,
- solving a parallel and asynchronous problem (CONCERTO vs AEOLUS).

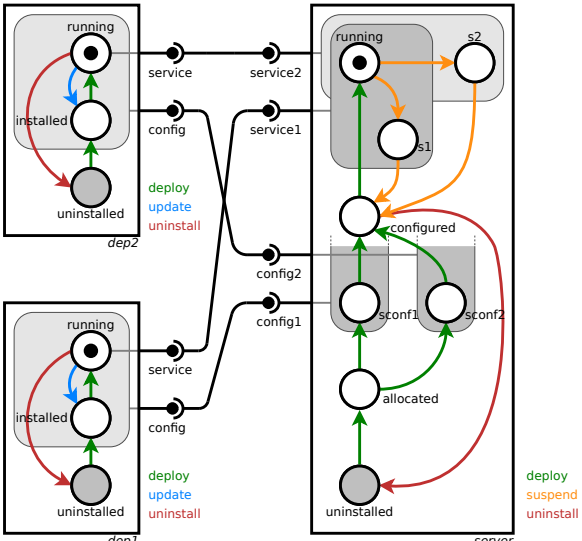
# Automated planning

## Automated planning

- **input:** a reconfiguration goal provided by the analysis phase
  - set of behaviors to execute on designated components
  - constraints on the final state of ports
- **output:** a reconfiguration script
  - behavior requests
  - synchronization commands
- out of scope: component creations/deletions/(dis)connections
  - usual for safety reasons to handle topological changes before and after behaviors requests and synchronization

1. find a solution for each component individually
2. find a global schedule for the assembly (SMT solver z3)
3. optimize synchronization barriers

# Example: updating components



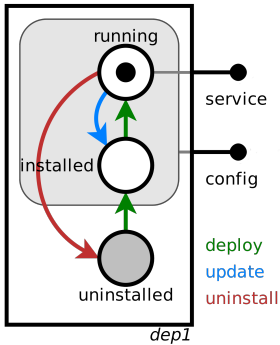

## Reconfiguration goal

1. `dep1` and `dep2` must execute `update`
2. all ports should be active at the end of the reconfiguration

**incomplete list of behaviors, a partial specification, is enough**

# 1. Individual component reconfiguration

- find a sequence of behaviors that satisfies the goal
- enumerate and analyze possible sequences
- no solution $\Longrightarrow$ failure of the procedure
- multiple solutions $\Longrightarrow$ pick one according to some optimization criterion
  - shortest sequence
  - shortest estimated execution time
  - fewest port requirements




**solution for this example**

[`update`, `deploy`]

# 2. Synchronized schedule

- **Goal**: find a global schedule
- **assumption**: reconfiguration plan with *steps*
  - at most 1 behavior per component/step
  - each step followed by global synchronization
- **problem: assign for each required behavior a given step to schedule it**
  - find function $schedule : Behavior \rightarrow Step$
  - find function $active_p : Step \rightarrow Bool$

```
pushB(server, suspend)
waitAll()
pushB(dep1, update)
pushB(dep2, update)
waitAll()
pushB(dep1, deploy)
pushB(dep2, deploy)
waitAll()
pushB(server, deploy)
waitAll()
```

What is expected from step 2.

# SMT encoding of the scheduling problem

- set of *Behavior* with one element for each behavior to schedule

**Example**

*dep*1.*deploy*, *dep*1.*update* etc.

- set of *Step* with elements $step_1, \ldots, step_n$
  - each behavior is executed is most 1 in a reconfiguration program

**Example**

maximum number of steps = 9 (number of behaviors)

- **constraints on ports and behaviors automatically extracted from the current Concerto configuration (assembly)**

# Constraints 1/2

Auxiliary functions $succ : Step \rightarrow Step$, $int : Step \rightarrow Nat$

- sequentiality of behaviors

$$int(schedule(b_1)) < int(schedule(b_2))$$

**Example**

from step 1. $int(schedule(dep1.update)) < int(schedule(dep1.deploy))$

- port requirements at the beginning of behaviors

$$active_p(schedule(b)) \qquad \neg active_u(schedule(b))$$

**Example**

$\neg active_u(schedule(dep1.update))$

# Constraints 2/2

- separation of behaviors with incompatible port effects

$$schedule(b) \neq schedule(b')$$

**Example**

$schedule(server.deploy) \neq schedule(dep1.update)$

- port status after behaviors

$$[\neg] active_p(succ(schedule(b)))$$

**Example**

$active_p(succ(schedule(dep1.deploy))$

# Missing port requirements

**Problem:** some unsatisfied ports requirements may make scheduling impossible

```
pushB(dep1, update)
pushB(dep2, update)
waitAll()
pushB(dep1, deploy)
pushB(dep2, deploy)
waitAll()
```

## Example

- we determined that `dep1` and `dep2` should execute `[update, deploy]`
- but the updates can't be executed while the server is relying on the provide ports

$$\neg active_u(schedule(dep1.update))$$

# Solving missing port requirements

**Solution:**

1. deduce new individual component goals
   - go to state that satisfies missing port requirement
   - go to state that satisfies final port constraints
2. extend the set of behaviors to schedule and go back to step 1

```
pushB(server, suspend)
waitAll()
pushB(dep1, update)
pushB(dep2, update)
waitAll()
pushB(dep1, deploy)
pushB(dep2, deploy)
waitAll()
pushB(server, deploy)
waitAll()
```

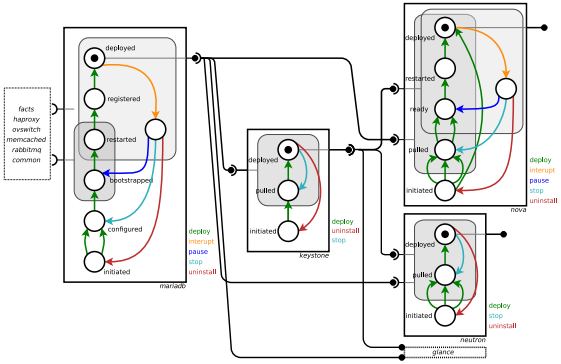
# 3. Synchronization optimization

```
pushB(server, suspend)
waitAll()
pushB(dep1, update)
pushB(dep2, update)
waitAll()
pushB(dep1, deploy)
pushB(dep2, deploy)
waitAll()
pushB(server, deploy)
waitAll()
```

```
pushB(server, suspend)
pushB(dep1, update)
pushB(dep2, update)
pushB(dep1, deploy)
pushB(dep2, deploy)
wait(dep1)
wait(dep2)
pushB(server, deploy)
wait(server)
```

1. replace global synchronization barriers by targetted ones
2. delay barriers whenever possible
   - behaviors with incompatible effects on connected ports must remain separated
   - no need to consider behavior pre-conditions: Concerto ensures implicit synchronization

# OpenStack-inspired use case




## Reconfiguration scenario

update database & restore system to working state

## Result

- reconfiguration script generated in 2.49 seconds
- contains only synchronization barriers that are needed
- 12 behaviors on 5 components

# Table of Contents

# Conclusion

- MAPE-K loop, (A) and (P) decision problems to automate further reconfiguration
- Satisfaction vs optimization problems
- State of the art: Zephyrus and Metis (Aeolus)
- CONCERTO program synthesis

**Ph.D. opportunity!**

- "Safe, efficient and low-energy self-adaptation for CyberPhysical Systems - Application to a scientific observatory in the Arctic tundra"
- Advisors: Hélène Coullon, Issam Raïs, Otto Anshus
- More details here

# References (SAT, SMT, CSP)

**[zephyrus1]** *Automated Synthesis and Deployment of Cloud Applications. Roberto Di Cosmo, Michael Lienhardt, Ralf Treinen. ASE 2014.*

**[zephyrus2]** *Zephyrus2: On the Fly Deployment Optimization Using SMT and CP Technologies. Erika Abraham, Florian Corzilius, Einar Broch Johnsen, Gereon Kremer.SETTA 2016.*

**[engage]** *Engage: A Deployment Management System. Jeffrey Fischer, Rupak Majumdar, Shahram Esmaeilsabzali. PLDI 2012.*

**[btrplace]** *BtrPlace: A Flexible Consolidation Managerfor Highly Available Applications. Fabien Hermenier, Julia Lawall, Gilles Muller. TDSC 2013.*

**[confsolve]** *A Declarative Approach to Automated Configuration. John A. Hewson, Paul Anderson, Andrew D. Gordon. LISA 2012.*

# References (heuristics)

**[metis]** *A Planning Tool Supporting the Deployment of Cloud Applications. Tudor A. Lascu, Jacopo Mauro, Gianluigi Zavattaro. ICTAI 2013.*

**[heuristic1]** *Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds. Pedro Silva, Christian Perez, Frédéric Desprez. CCGrid 2016.*

**[heuristic2]** *Online Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization. Emile Cadorel, Hélène Coullon, Jean-Marc Menaud. CCGrid 2020.*

# Questions?