Towards Management of Unsatisfiable Reconfiguration in Ballet

Jolan Philippe, Hélène Coullon, Charles Prud'Homme

October 7th, 2024

DiverSE team - IRISA

Hello world

Jolan Philippe, Postdoc in the Taranis project, WP2
Université de Rennes, DiverSE Team
(λx.λy.x@y) Jolan.Philippe inria.fr

Topics of interest

- Distributed computing
- Model Driven Engineering
- Reconfiguration in Fog and Cloud environment

More details on: `https://jolanphilippe.github.io/`

DevOps deployment and reconfiguration

Responsible for **development**

DevOps team

Run tests → **Package app** → **Build image Docker** → **Push artifact to repo** → **Deploy to server**

Continuous integration + Continuous deployment

Responsible for **operations**

Cross-DevOps reconfiguration

Responsible for **development** of A

DevOps team

Run tests on A → Package A → Build image for A → Push A to repo → Deploy A

Responsible for **operations** on A

Responsible for **development** of B which uses A

DevOps team

Run tests on B → Package B → Build image for B → Push B to repo → Deploy B (after A?)

Responsible for **operations** on B (using A)

Ballet for cross-DevOps reconfiguration

Ballet

- DevOps declarative tool (YAML) for decentralized reconfiguration
- One instance of Ballet per node, each managed by one DevOps team
- Inspired by Concerto, an execution model combining CBSE with state machine
- Components defined with fine-grained lifecycles for control components, and ports for dependencies
- A two-phase approach: Planning and Execution

Ballet's usage: Developer's concern

Life-cycle and dependencies

Simple language to define component

- **Places**: milestones of the reconfiguration
- **Behaviors**: interface of actions for the DevOps
- **Transitions**: concrete actions between places, associated to behaviors
- **Ports**: Provide (resp. use) information to (resp. from) external components
  - Ports are bounded to places and transitions

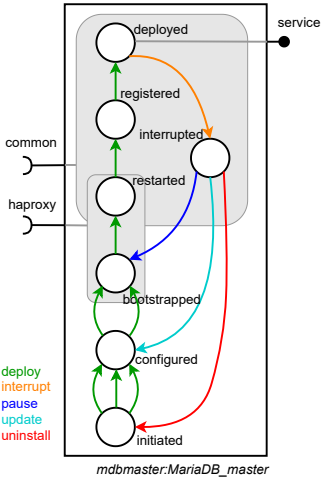


Figure 1: Visual representation of a component for MariaDB

Ballet's usage: Developer's concern

Listing 1: Control component MariaDB master in PYTHON

```python
class MariaDB_Master(Component):
    def create(self):
        self.places = [ "initiated", "configured", "bootstrapped", "restarted",
                        "registered", "deployed", "interrupted"]
        self.transitions = {
            "configure0": ("initiated", "configured", "deploy", self.configure0),
            "configure1": ("initiated", "configured", "deploy", self.configure1),
            "configure2": ("initiated", "configured", "deploy", self.configure2),
            ...
        }
        self.dependencies = {
            "service": (DepType.PROVIDE, ["deployed"]),
            "haproxy": (DepType.USE, ["bootstrapped","restarted"]),
            ...
        }
        self.initial_place = 'initiated'
        self.running_place = 'deployed'

    def configure0(self):
      # concrete actions
```

Ballet's usage: DevOps' concern

Target assembly (YAML)

- A list of components to appear
- How components are connected

Reconfiguration goals

Declarative language for defining reconfiguration goals

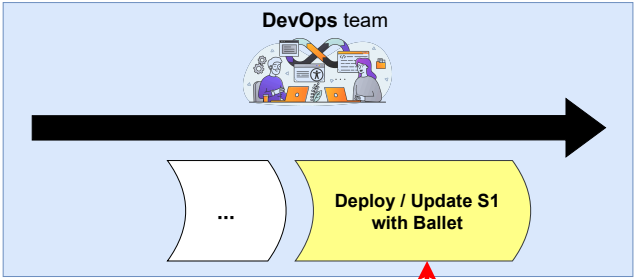
- **Behavior goal**: Specify a behavior that must be executed
- **Port goal**: Specify a port status (active, inactive)
- **State goal**: Specify a component state (specific, running, initial)

Listing 2: Language to define reconfiguration goals for DevOps usage

```
<goals> ::= behaviors: <bhvr_list>
            ports: <port_list>
            components: <comp_list>
<bhvr_list> ::= ...
<bhvr_item> ::= - forall: <bhvr_name>
              | - component: <comp_name>
                  behavior: <bhvr_name>
<port_list> ::= ...
<port_item> ::= - forall: <port_status>
              | - component: <comp_name>
                  port: <port_name>
                  status: <port_status>
<comp_list> ::= ...
<comp_item> ::= - forall: <comp_status>
              | - component: <comp_name>
                  status: <comp_status>
```

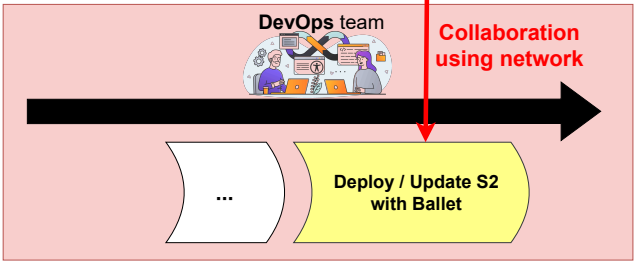
Ballet for cross-DevOps reconfiguration

Responsible for **development** of S1



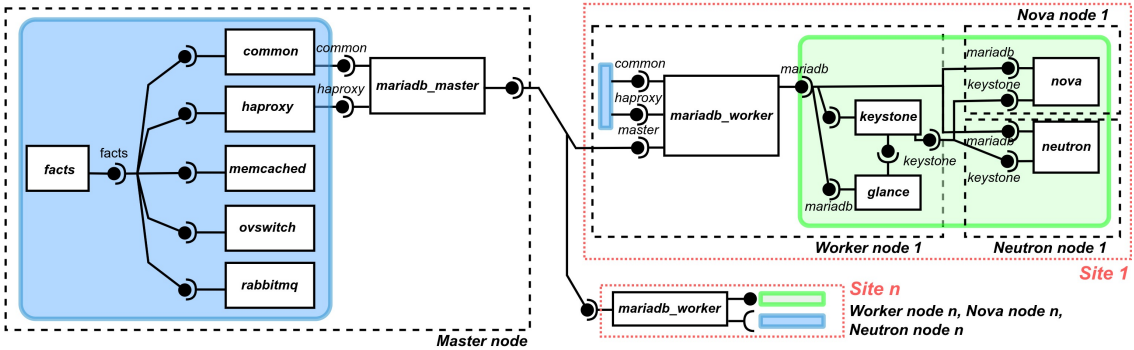
Responsible for **operations** on S1

Responsible for **development** of S2 which uses S1



Responsible for **operations** on S2 (using S1)

Example: Update a multi-site OpenStack with Galera cluster of MariaDB



Example: Update a multi-site OpenStack with Galera cluster of MariaDB

```
components:
  - mariadb_master:
      type: 'MariaDB_master'
  - facts:
      type: 'Facts'
  - common:
      type: 'Common'
  - haproxy:
      type: 'HAProxy'
  - ...
```

```
goals:
  - behaviors:
      component: mariadb_master
      behavior: update
  - components:
      forall: initial
```

```
connections:
  - facts,service,common,facts_service
  - facts,service,haproxy,facts_service
  - ...
  ${{ each site in range(0,sites) }}:
    - mariadb_master,service,mdbworker${{ site }},master_service
```

Executing a reconfiguration using Ballet

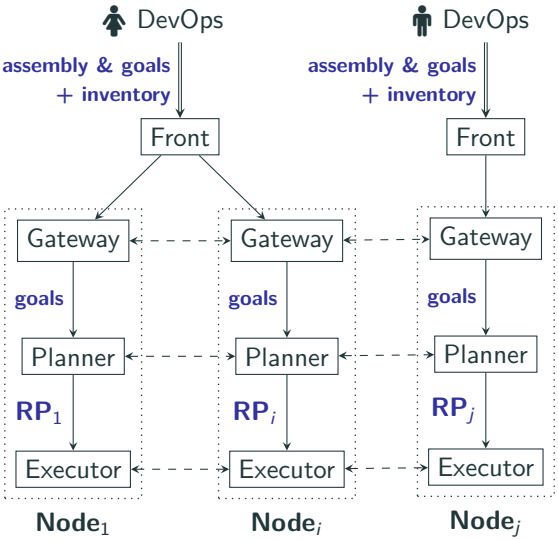


Figure 2: Ballet overview

Planner

Decentralized inference of reconfiguration plans (RPs)

1. Make a diff with current status: components to add/remove and (dis)connect
2. Iterative process for local behaviors
   - Local inference of behaviors
   - Constraint propagation (Gossip)
   - When propagation ends, inference of RPs

Executor [A. Omond's thesis]

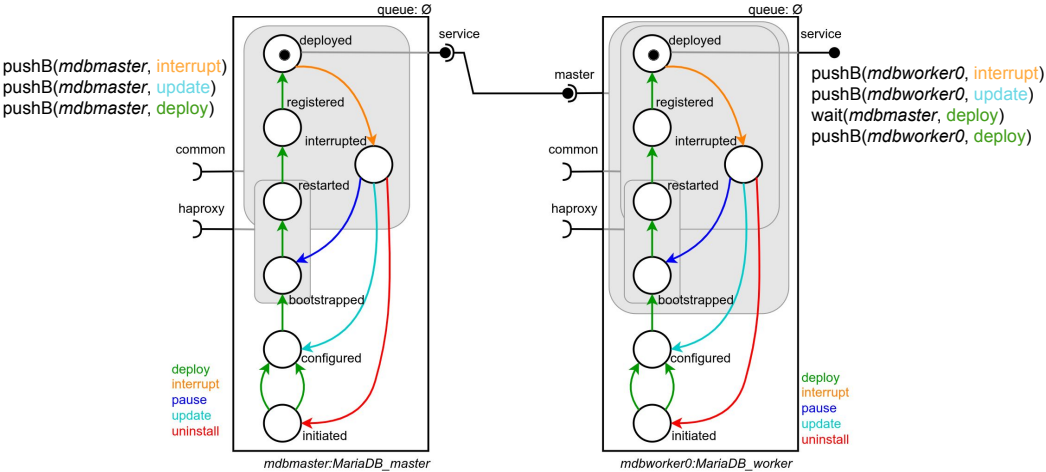
Coordinated execution of RPs

Reconfiguration execution in Ballet

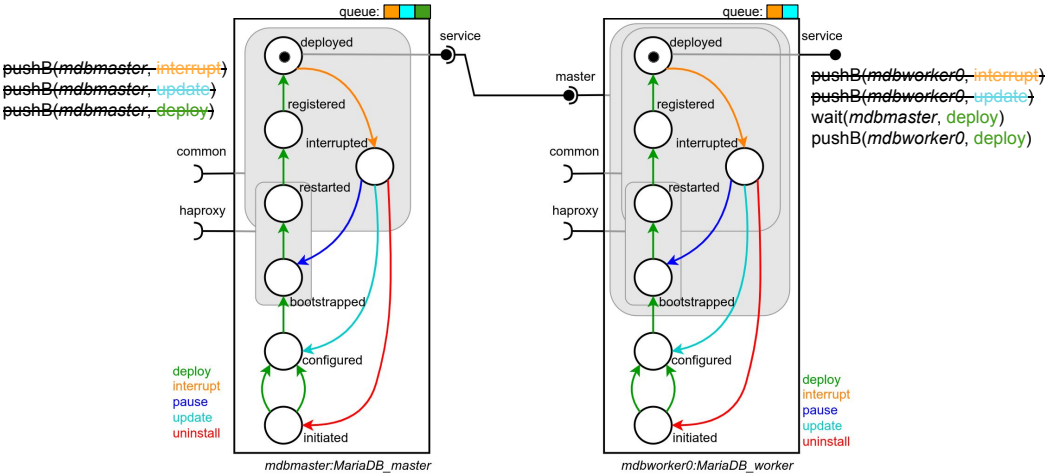
Concerto-D

- Decentralized execution for Concerto model
- Small language
  - `add`(id_c, t_c) / `del`(id_c):
    add/remove a component instance to the current assembly
  - `con`(id_{c_1}, u, id_{c_2}, p) / `discon`(id_{c_1}, u, id_{c_2}, p):
    connect/disconnect two component instances with compatible ports
  - `pushB`(id_c, b):
    push behavior to the behavior queue on a component instance
  - `wait`(id_c, b):
    wait for a given component instance to execute a behavior

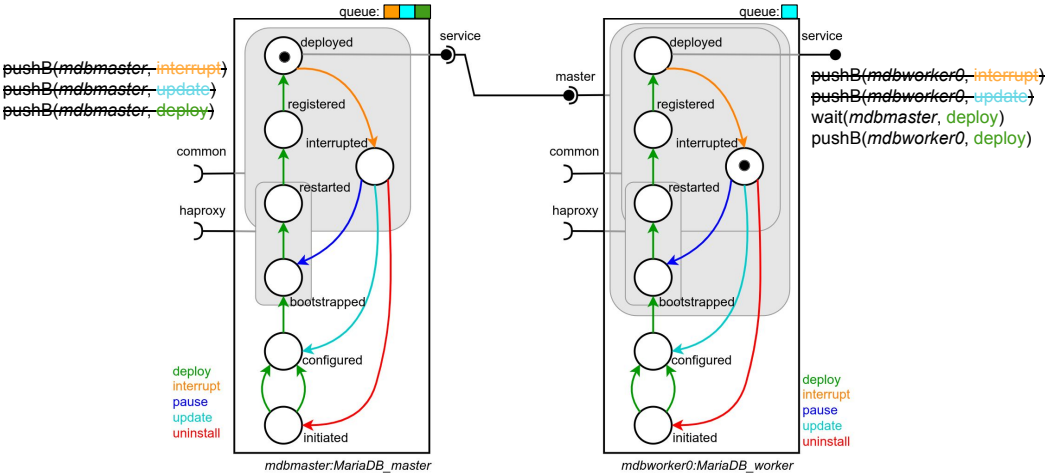
Reconfiguration execution in Ballet



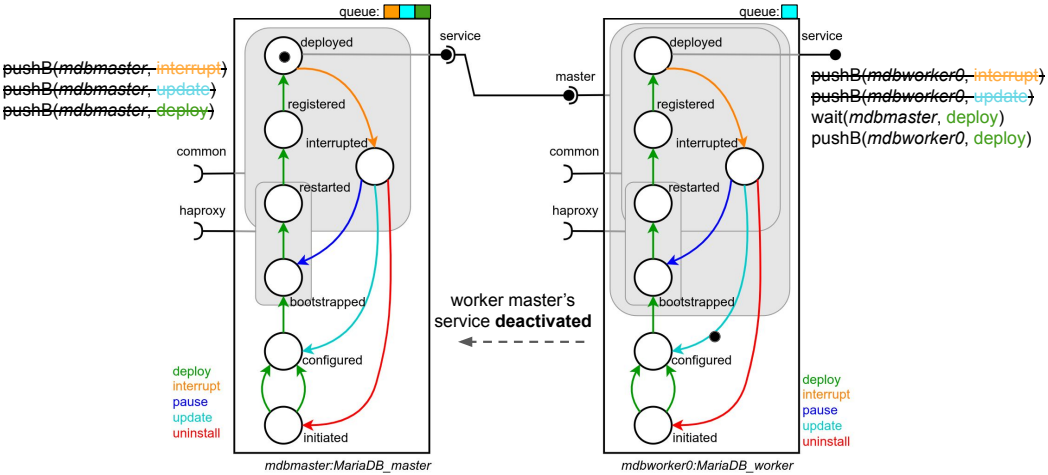
Reconfiguration execution in Ballet



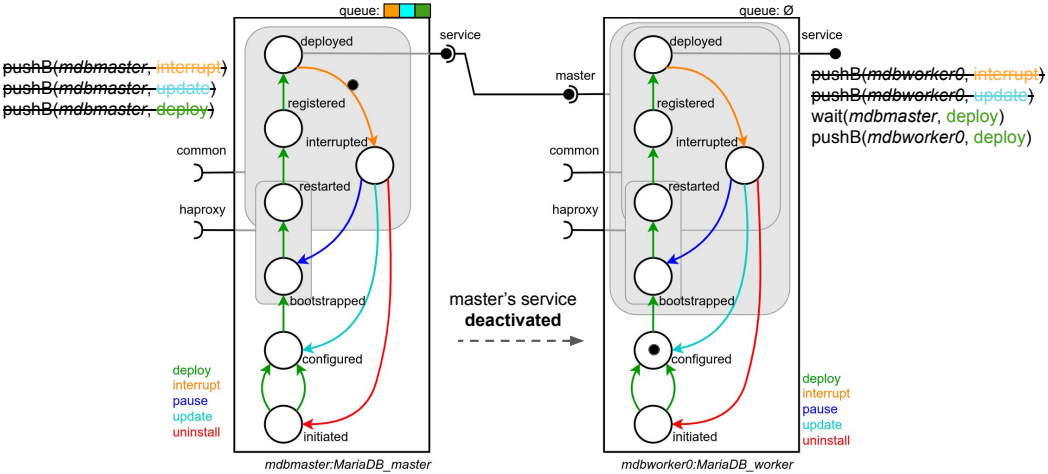
Reconfiguration execution in Ballet



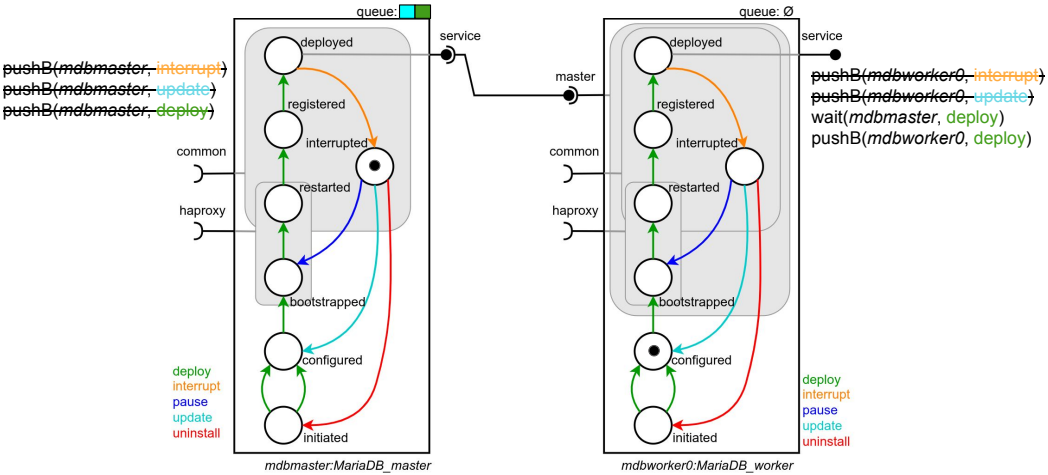
Reconfiguration execution in Ballet



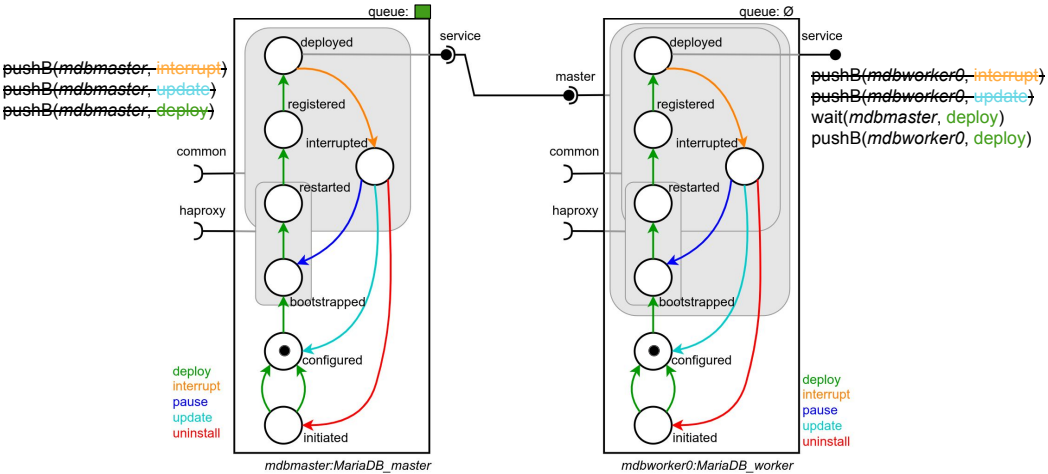
Reconfiguration execution in Ballet



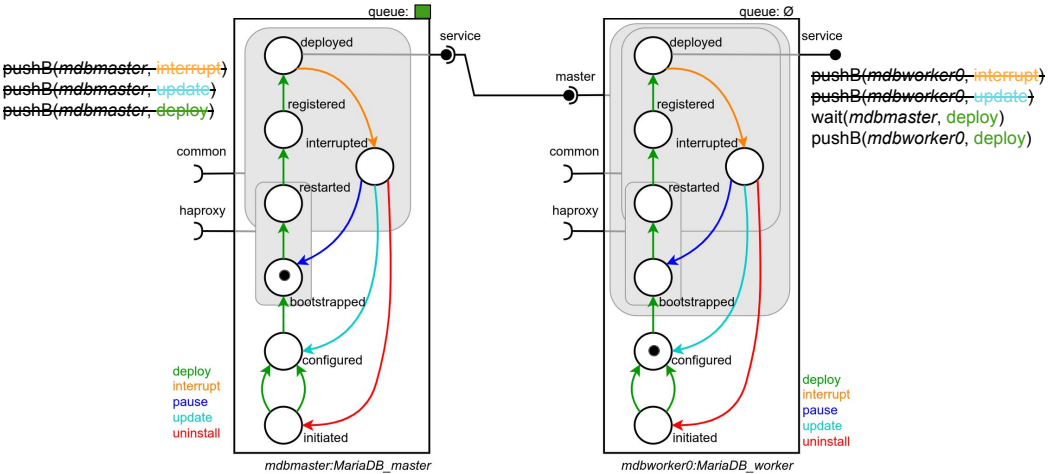
Reconfiguration execution in Ballet



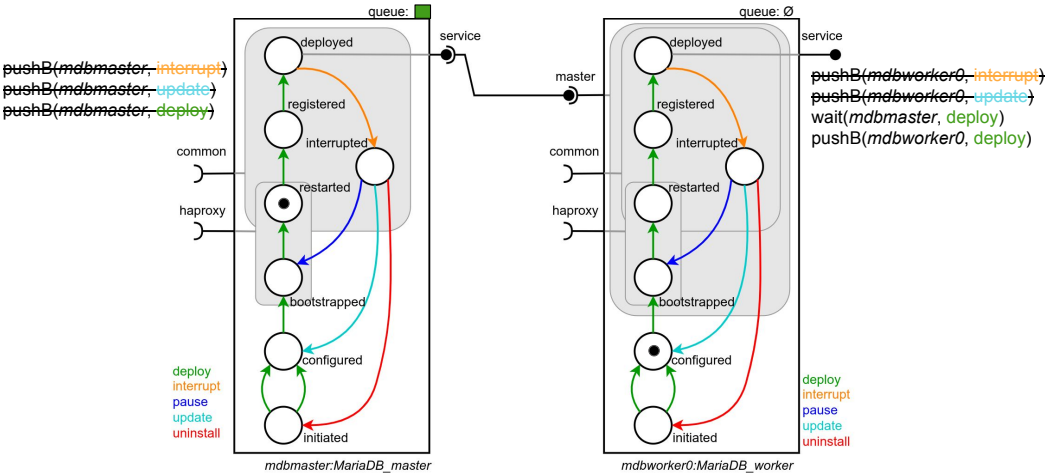
Reconfiguration execution in Ballet



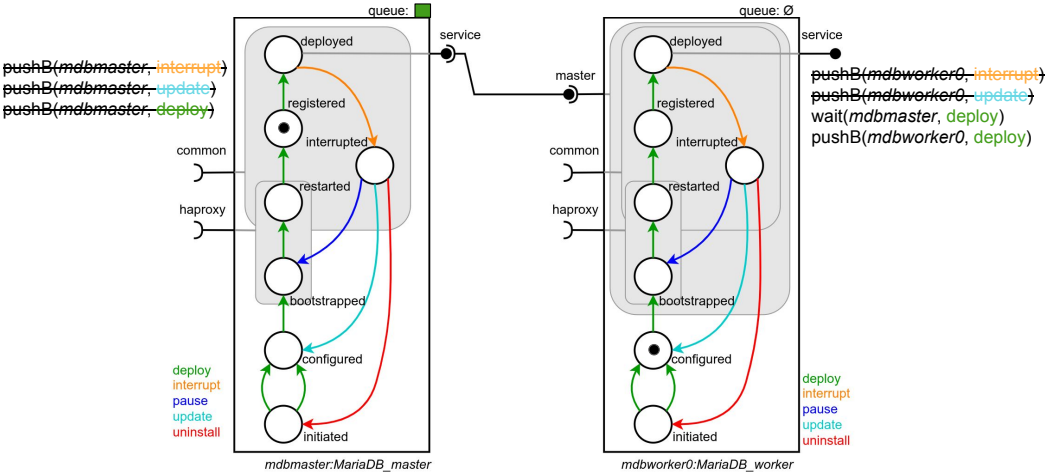
Reconfiguration execution in Ballet



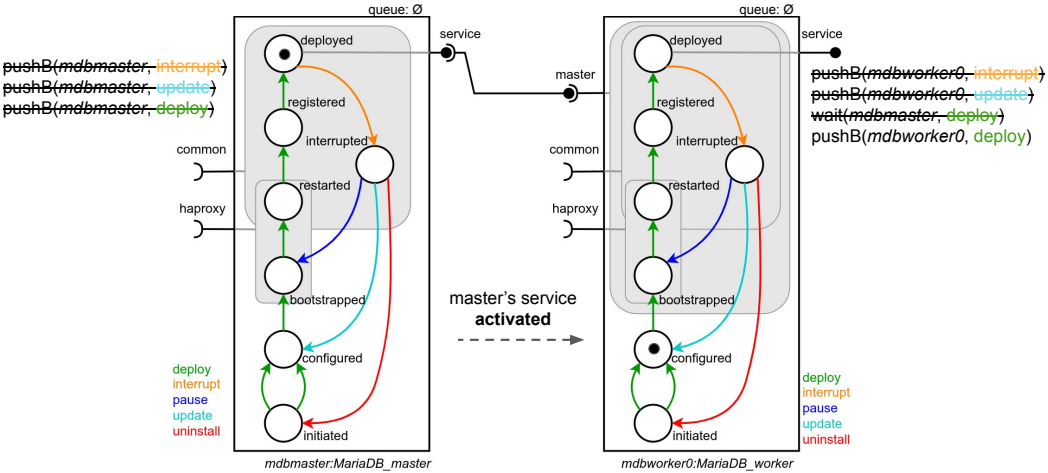
Reconfiguration execution in Ballet



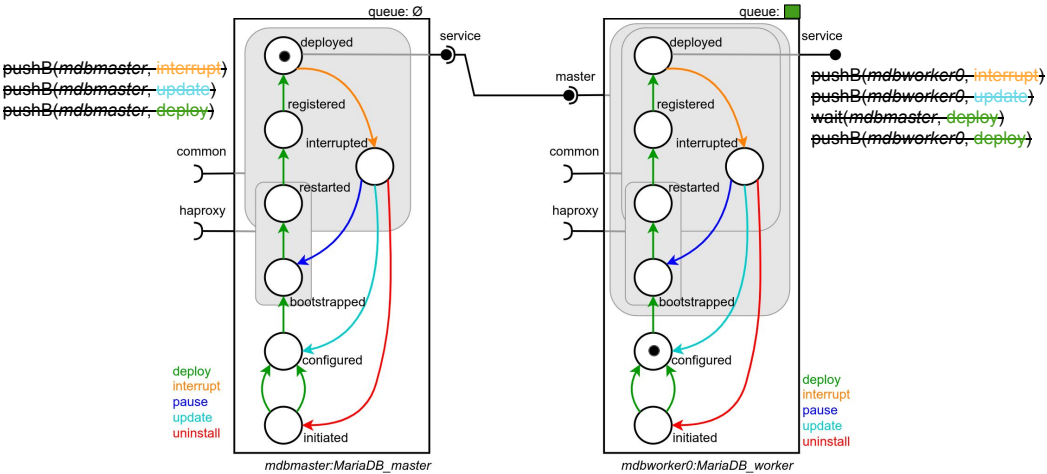
Reconfiguration execution in Ballet



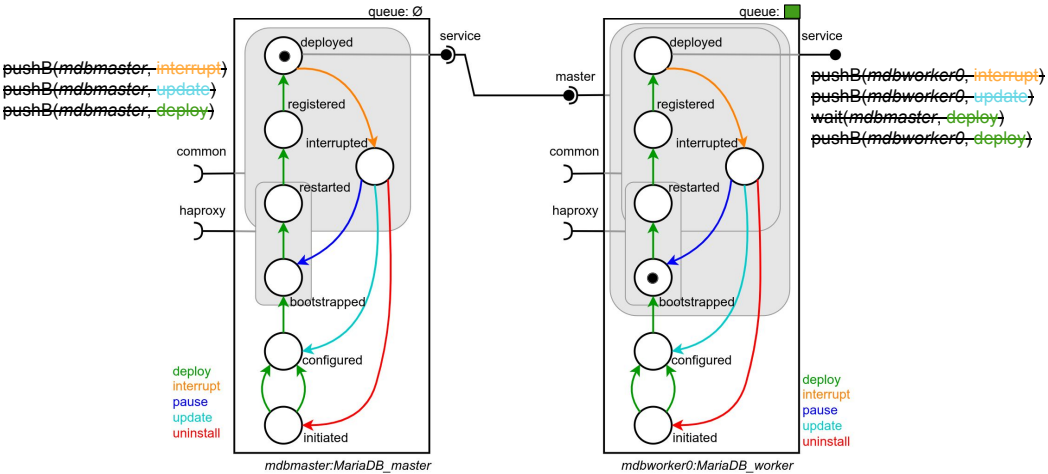
Reconfiguration execution in Ballet



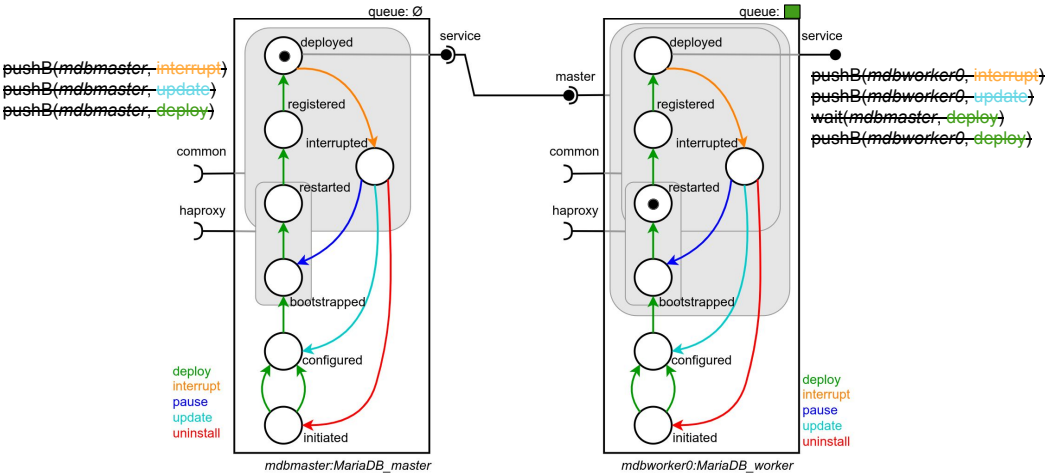
Reconfiguration execution in Ballet



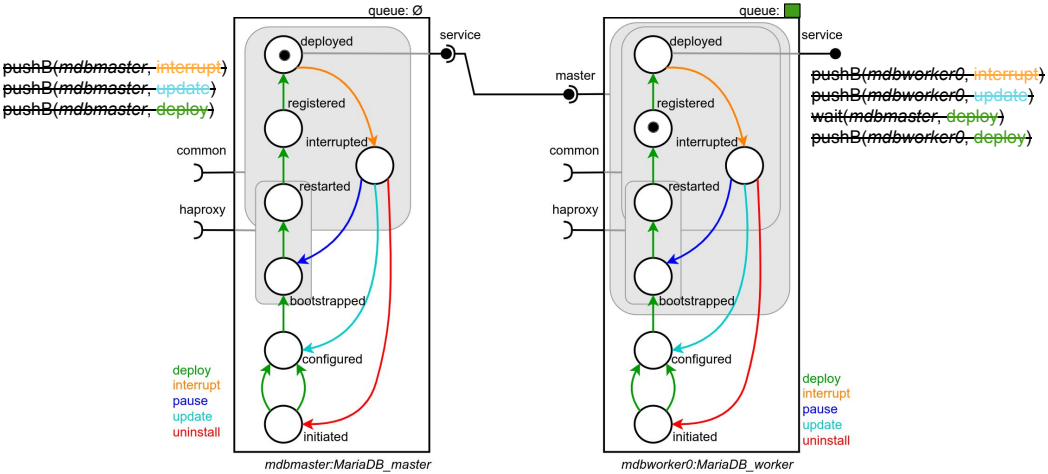
Reconfiguration execution in Ballet



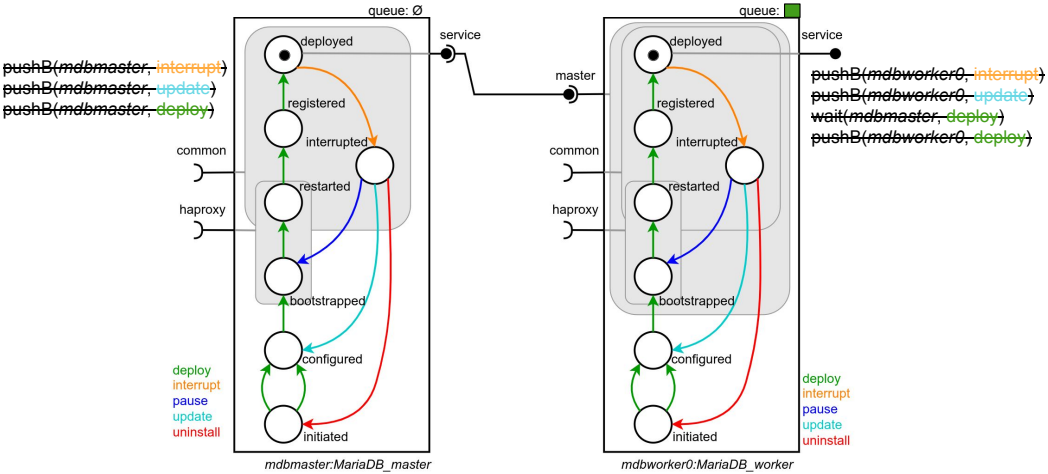
Reconfiguration execution in Ballet



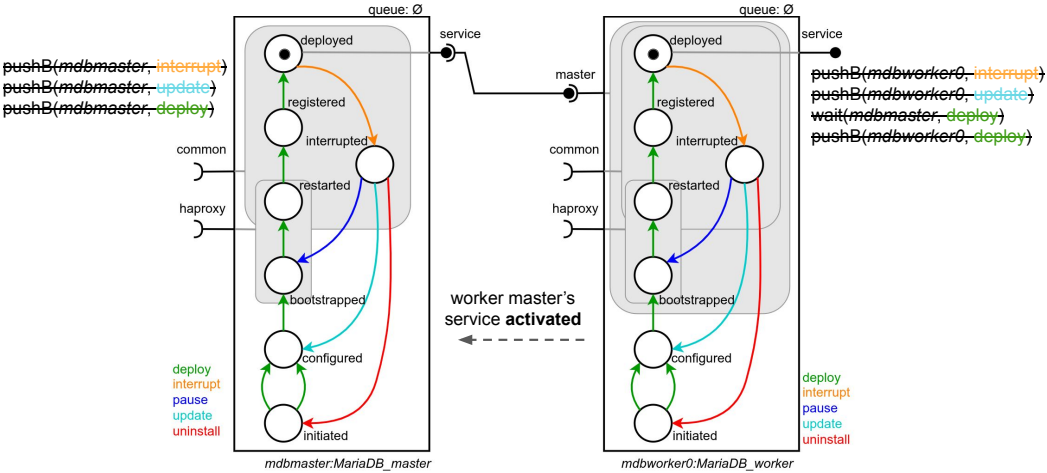
Reconfiguration execution in Ballet



Reconfiguration execution in Ballet



Reconfiguration execution in Ballet



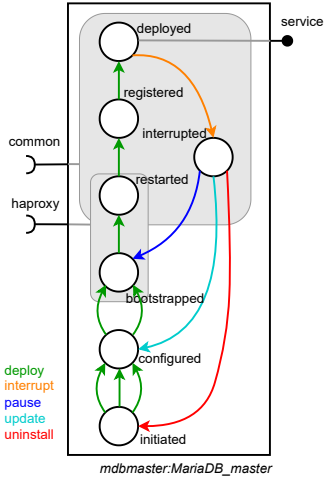
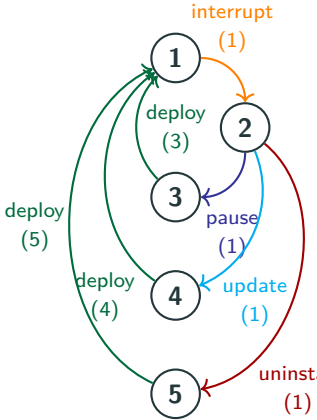


Figure 3: *MariaDB_master* control component



state	1	2	3	4	5
common	✓	✓	×	×	×
haproxy	×	×	✓	×	×
service	✓	×	×	×	×

Figure 4: Automaton representation of *Mariadb_master* component's life cycle with its matrix for ports statuses.

Unsatisfiable reconfiguration

Example 1

A service used by 2 other services is now deprecated

- One DevOps team decides to remove it, and replace it with another one
- One DevOps team decides to keep it on, and update it

Example 2

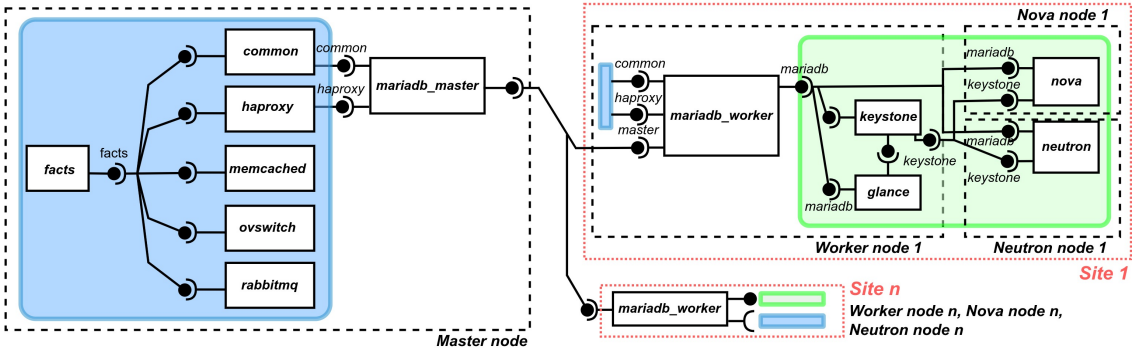
The current version 1.0 of a database in a master-worker architecture is not maintained anymore. The master must be updated, and also the workers accordingly

- One DevOps team decides to update the master to 1.2
- One DevOps team decides to update a worker 2.0

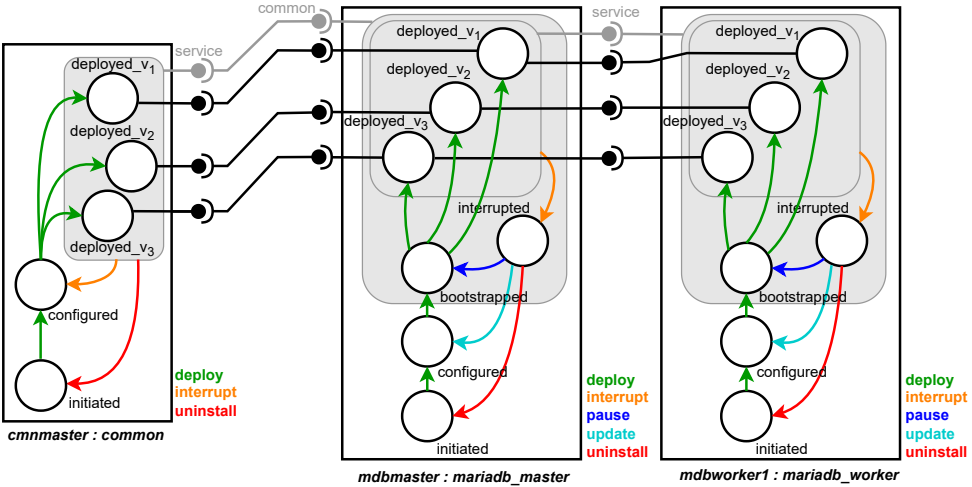
Running example

Reconfiguration in multi-site OpenStack such as

$\forall\ c1 : Component, \forall\ c2 : Component,\ \ compatible(c1.version, c2.version)$

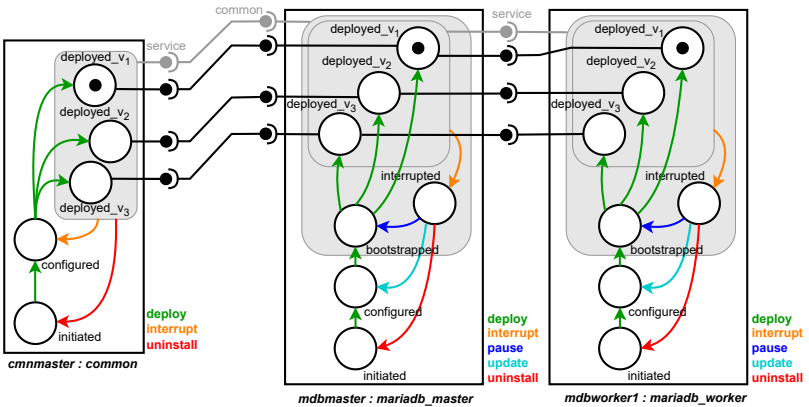


Versioned components in OpenStack



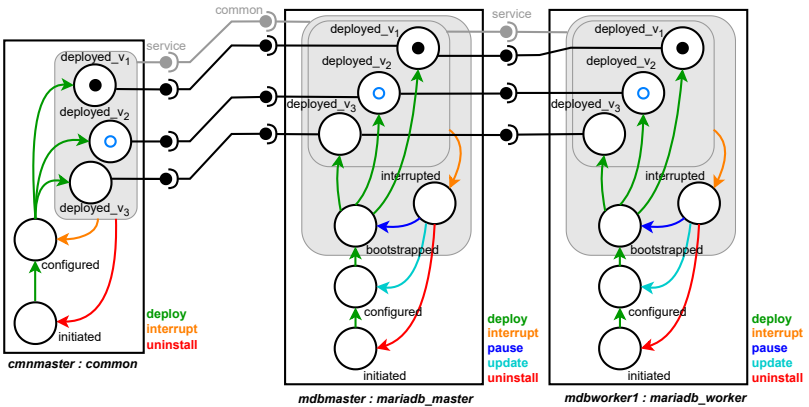
Satisfiable reconfiguration

```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```



Satisfiable reconfiguration

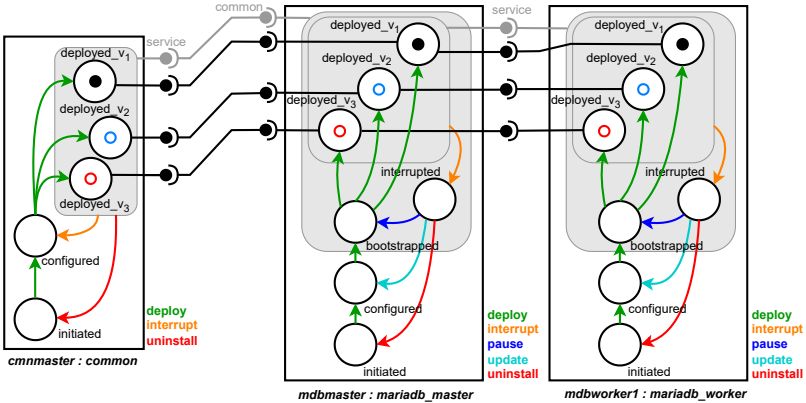
```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```



Unsatisfiable reconfiguration

```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```

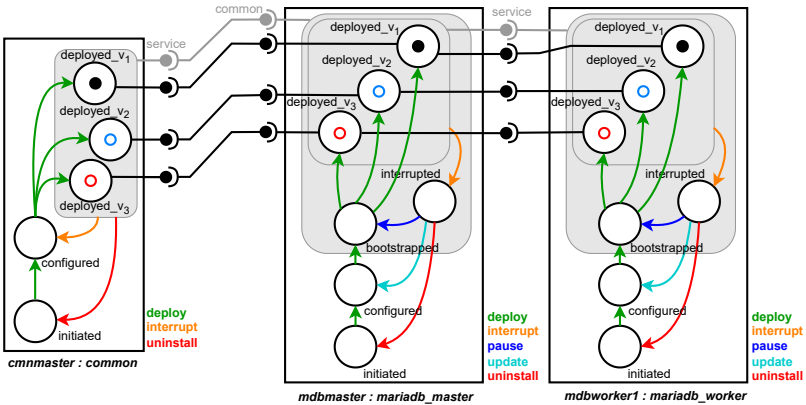
```
goals:
- components:
  - component: mdbworker1
    status: deployed_v3
- ports:
  - forall
    port: service
    status: active
```



Unsatisfiable reconfiguration

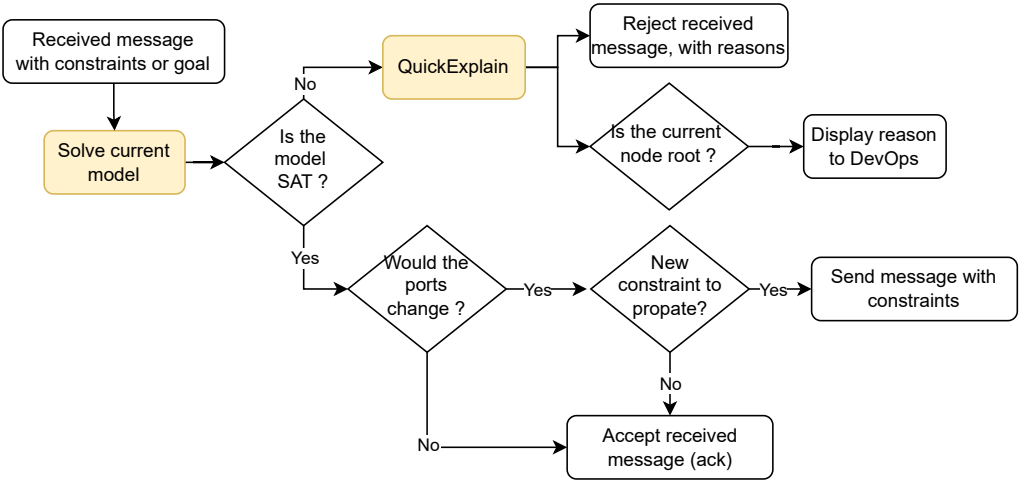
```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```

```
goals:
- components:
  - component: mdbworker1
    status: deployed_v3
- ports:
  - forall
    port: service
    status: active
```



⇒ Unsatisfiable goals

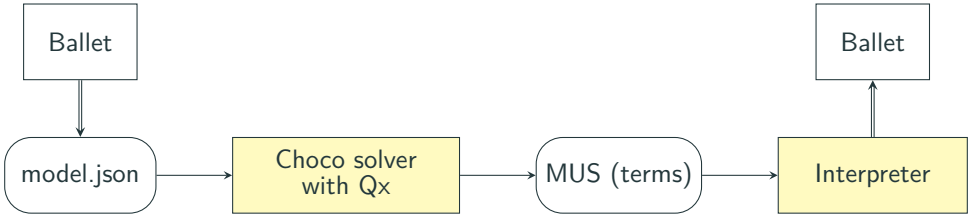
Unsatisfiable management



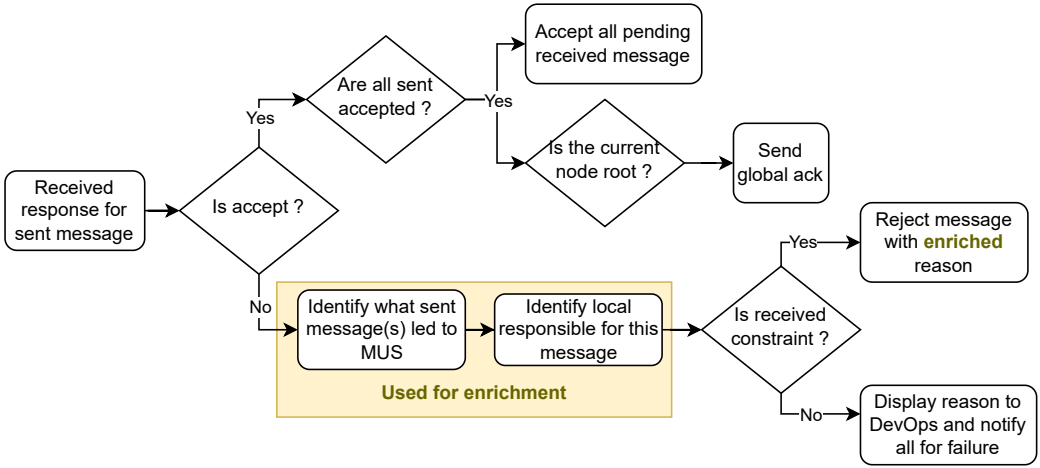
Explainability

Ballet using QuickExplain (Qx)

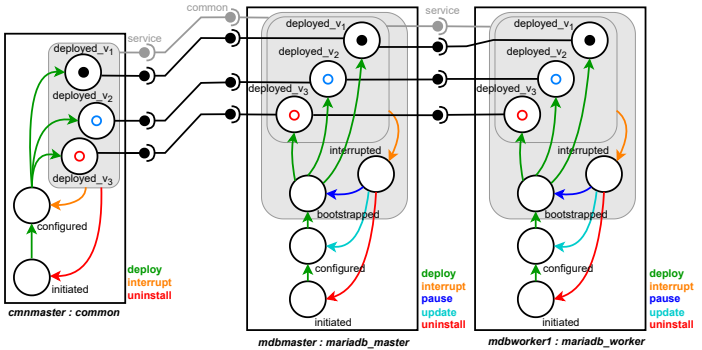
- Make a JSON file with constraints
- Load constraints in Choco solver, and annotate all CP terms with concrete constraint
- Use Qx dichotomy algorithm to get minimum unsat set (MUS) of terms
- Terms mapped with concrete constraints



Backtracking for explainability

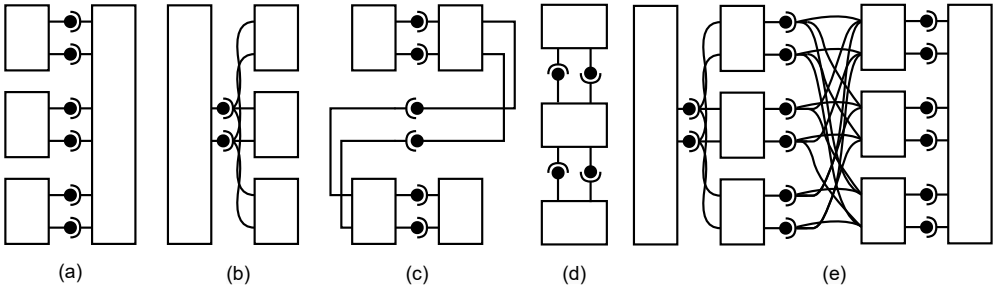


Output for unsatisfiable reconfiguration




```
CMP: mdbmaster's common_v2 <=> deployed_v2
CMP: mdbmaster's service_v3 <=> deployed_v3
GOAL from DevOps1: cmnmaster's deployed_v2 => cmnmaster's service_v2
    =INF=> mdbmaster's common_v2
GOAL from DevOps2: mdbworker1's deployed_v3 => mdbworker1's master_v3
    =INF=> mdbmaster's service_v3
```

Tests on topological examples



Results

- It works on the 5 topologies (many-to-one, one-to-many, linear, circular, stratified)
- Tested on a multi-OpenStack case assembly with versioned component
- Small overhead for explainability

Conclusion

Contributions

- UNSAT detection at planning time for decentralized reconfiguration
- Extended Choco solver with an annotation mechanism for explainability
- Integrated to Ballet reconfiguration tool

More about Ballet

[SANER24] Jolan Philippe, Antoine Omond, Hélène Coullon, Charles Prud'Homme, and Issam Raïs. Fast Choreography of Cross-DevOps Reconfiguration with Ballet: A Multi-Site OpenStack Case Study.

[ICE24] Farid Arfi, Hélène Coullon, Frédéric Loulergue, Jolan Philippe, and Simon Robillard. A Maude Formalization of the Distributed Reconfiguration Language Concerto-D.

Perspectives

- Conduct experiments on real case scenario
- Suggestion for conflict resolution if possible