Decentralized reconfiguration plan synthesis

Jolan PHILIPPE

PostDoc - SeMaFoR project

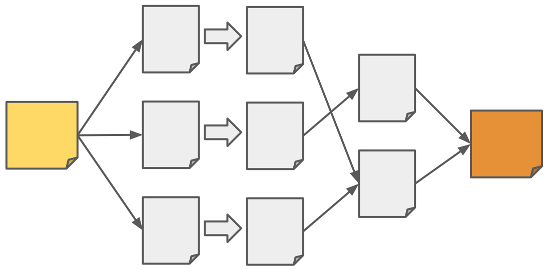
20th April 2023

Resume

NAU
SSERL

Master's degree

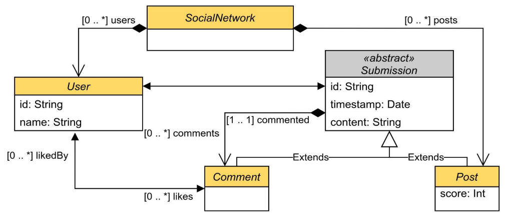
- Parallel programming and skeletons
- Correctness of programs
- Distributed computing (MPI)



IMT Atlantique
naomod

Ph.D

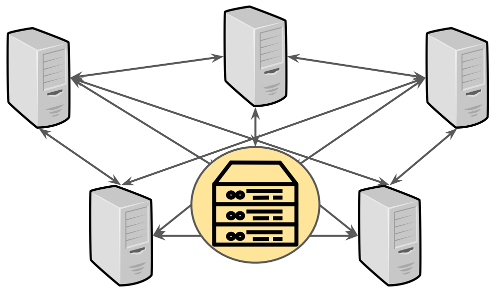
- Model-Driven Engineering
- Distributed computing
- Feature analysis



IMT Atlantique
STACK

Postdoc

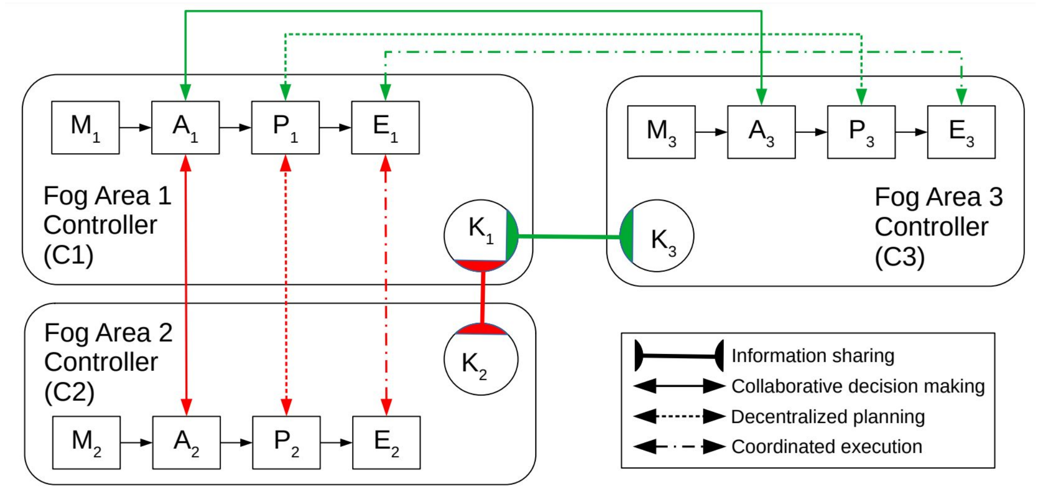
- Fog computing
- (Re)configuration of systems
- Constraint programming



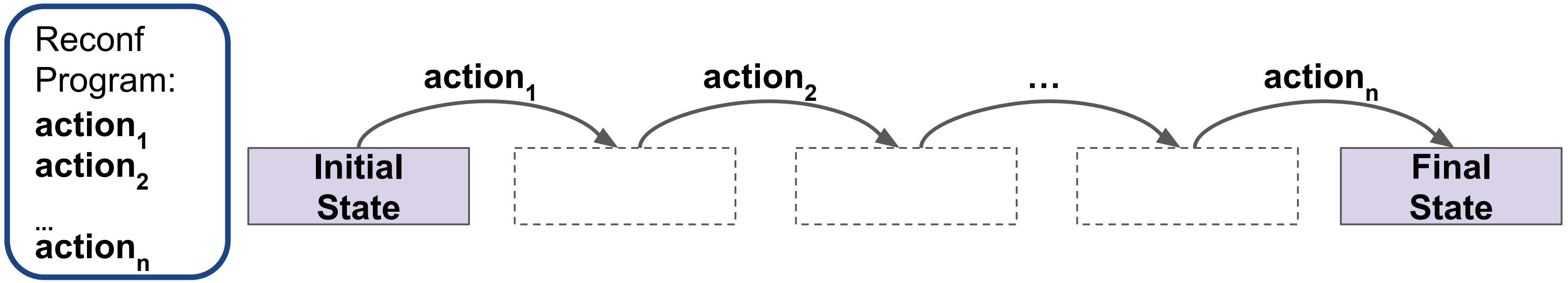
MAPE-K and SeMaFoR

Coordinated Control Pattern model

- **M**onitor its state and the state of the environment
- **A**nalyze to decide which state to reach → **WP2**
- **P**lan the reconfiguration → **WP3**
- **E**xecute the reconfiguration to reach the new state
  - **K**nowledge that is common, to take a decision


M1
A1
P1
E1
Fog Area 1 Controller (C1)
K1
M3
A3
P3
E3
Fog Area 3 Controller (C3)
K3
Fog Area 2 Controller (C2)
K2
M2
A2
P2
E2
Information sharing
Collaborative decision making
Decentralized planning
Coordinated execution

Reconfiguration planning



Objectives:

➢ Infer reconfiguration actions

➢ Optimal overall reconfiguration

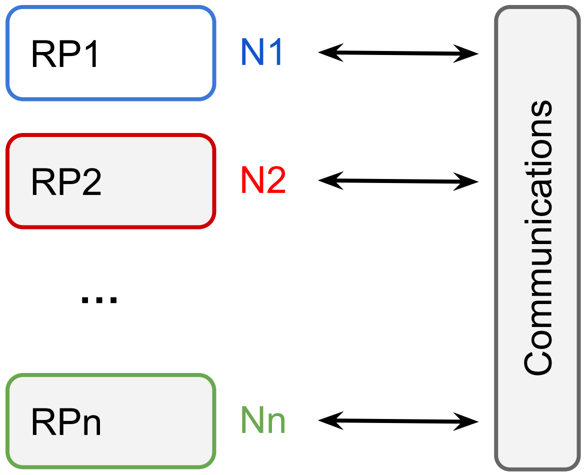
Challenges:

■ Locally: partial view of the system

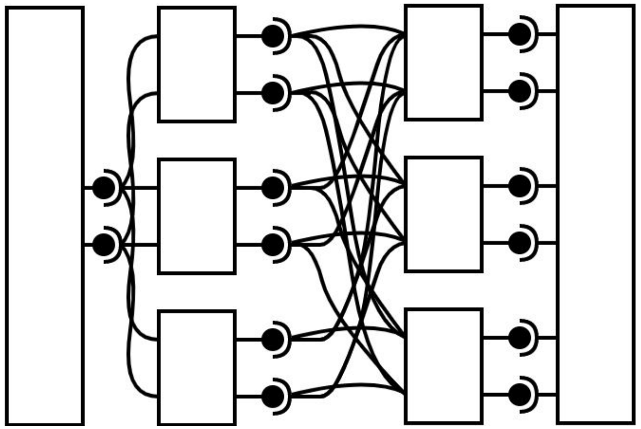
■ Collaboration with neighborhood

Inspiration:

■ SMT-based work by Robillard et. al.



Concerto-D (Antoine Omond's PhD)



Components are connected using ports:

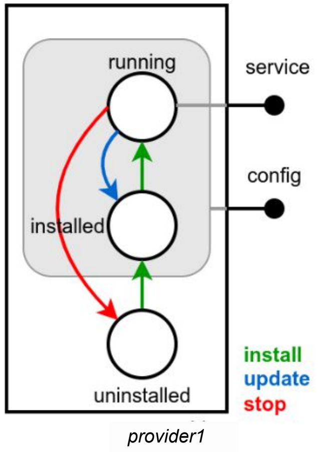
- **Provide port**
- **Use port**

creating coordination constraints

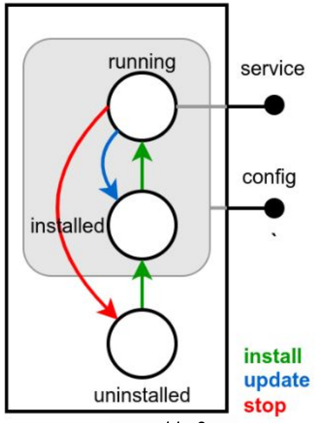
Concerto-D: A reconfiguration language for decentralized components

- Involved components
- Interactions / connections between components
- Changes in the component

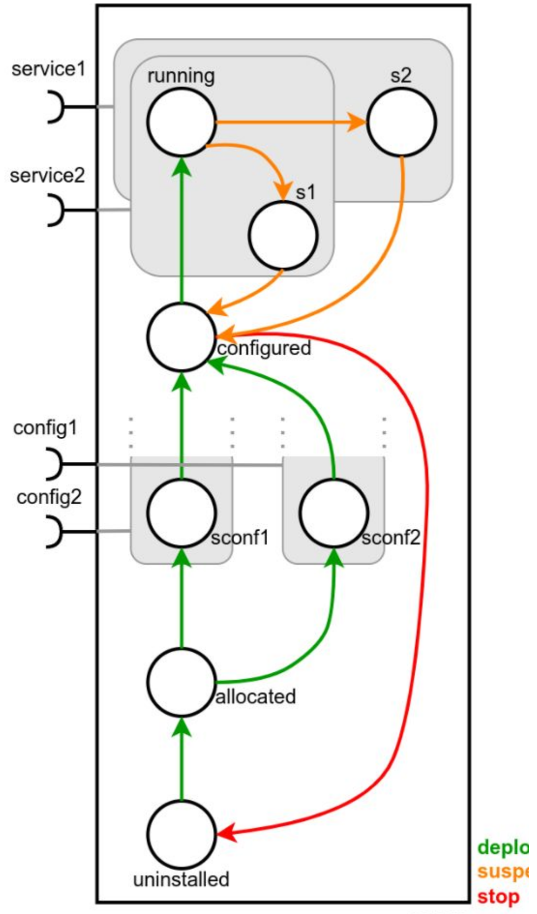
Concerto-D: Involved components



provider1



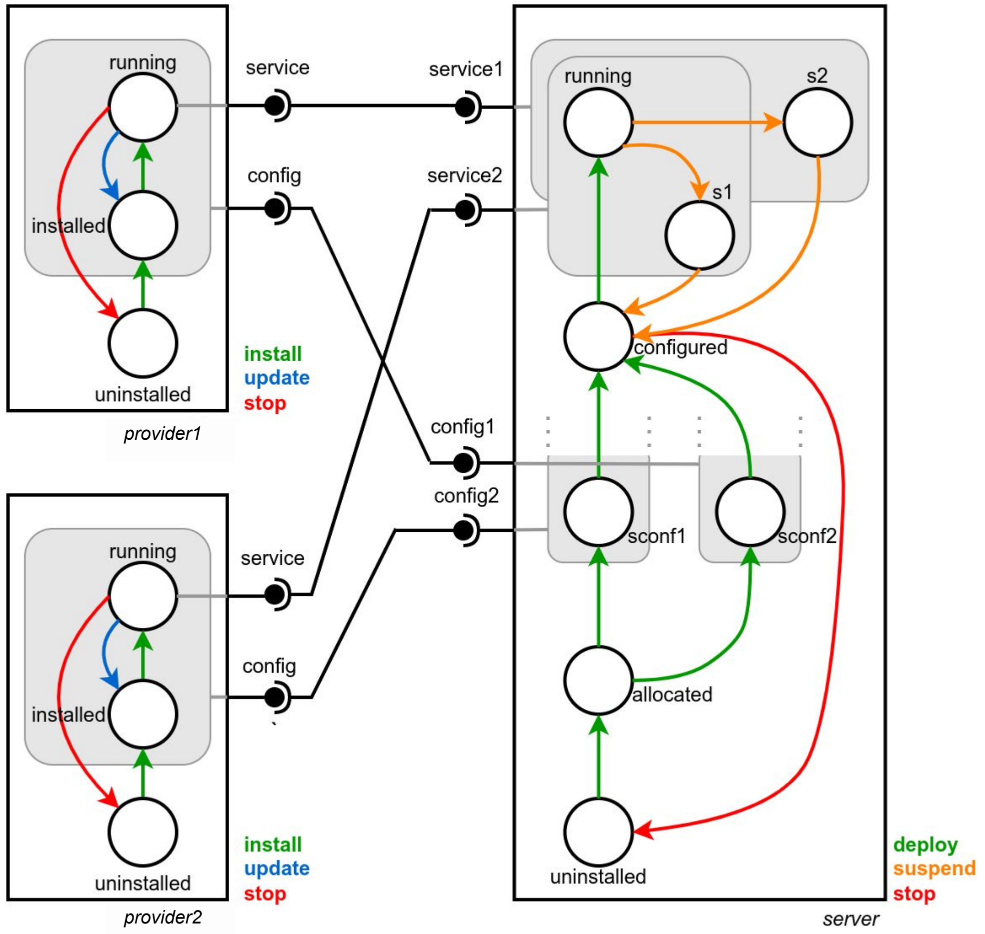
provider2



server

add("*provider1*", Provider)
add("*provider2*", Provider)
add("*server*", Server)

Concerto-D: Connections between components

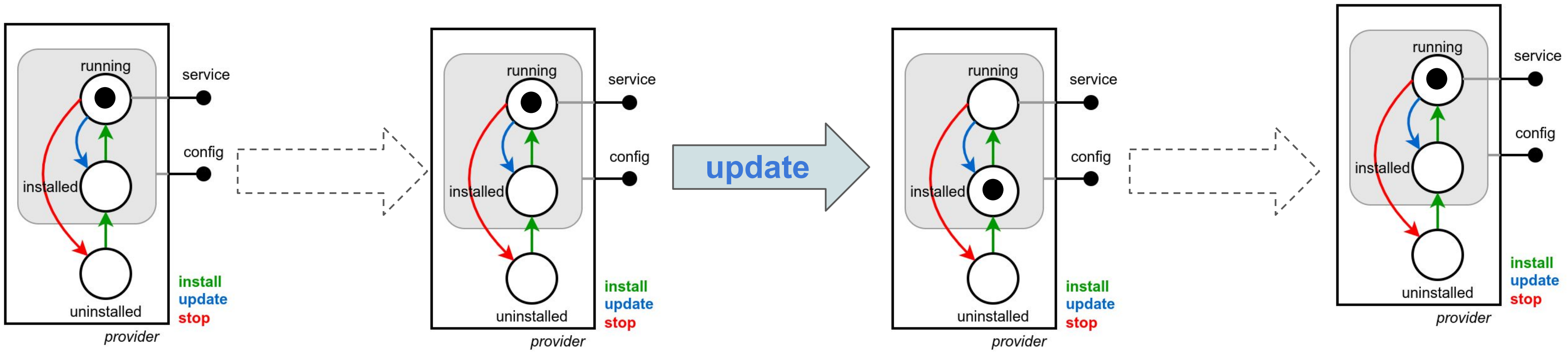


add("*provider1*", Provider)
add("*provider2*", Provider)
add("*server*", Server)
connect("provider1", "service",
  "server", "service1")
connect("provider1", "config",
  "server", "config1")
connect("provider2", "service",
  "server", "service2")
connect("provider2", "config",
  "server", "config2")

Concerto-D: State and changes in the component

Example of objective:

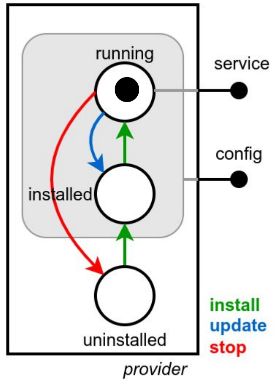
- Update a running *provider*
- End the reconfiguration with a running *provider*

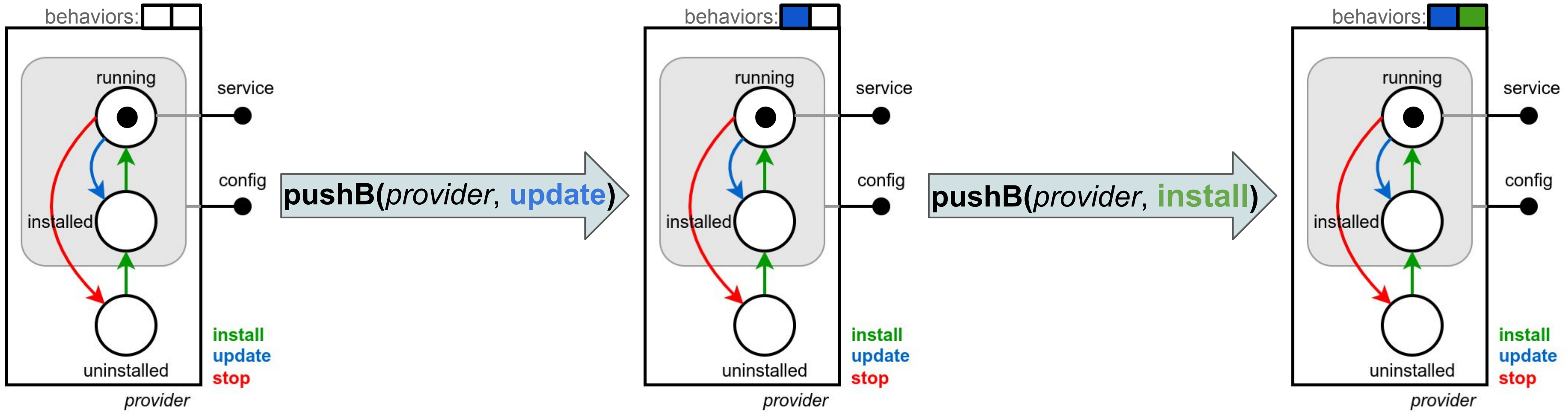


Inferred actions:
- update *provider*
- install *provider*

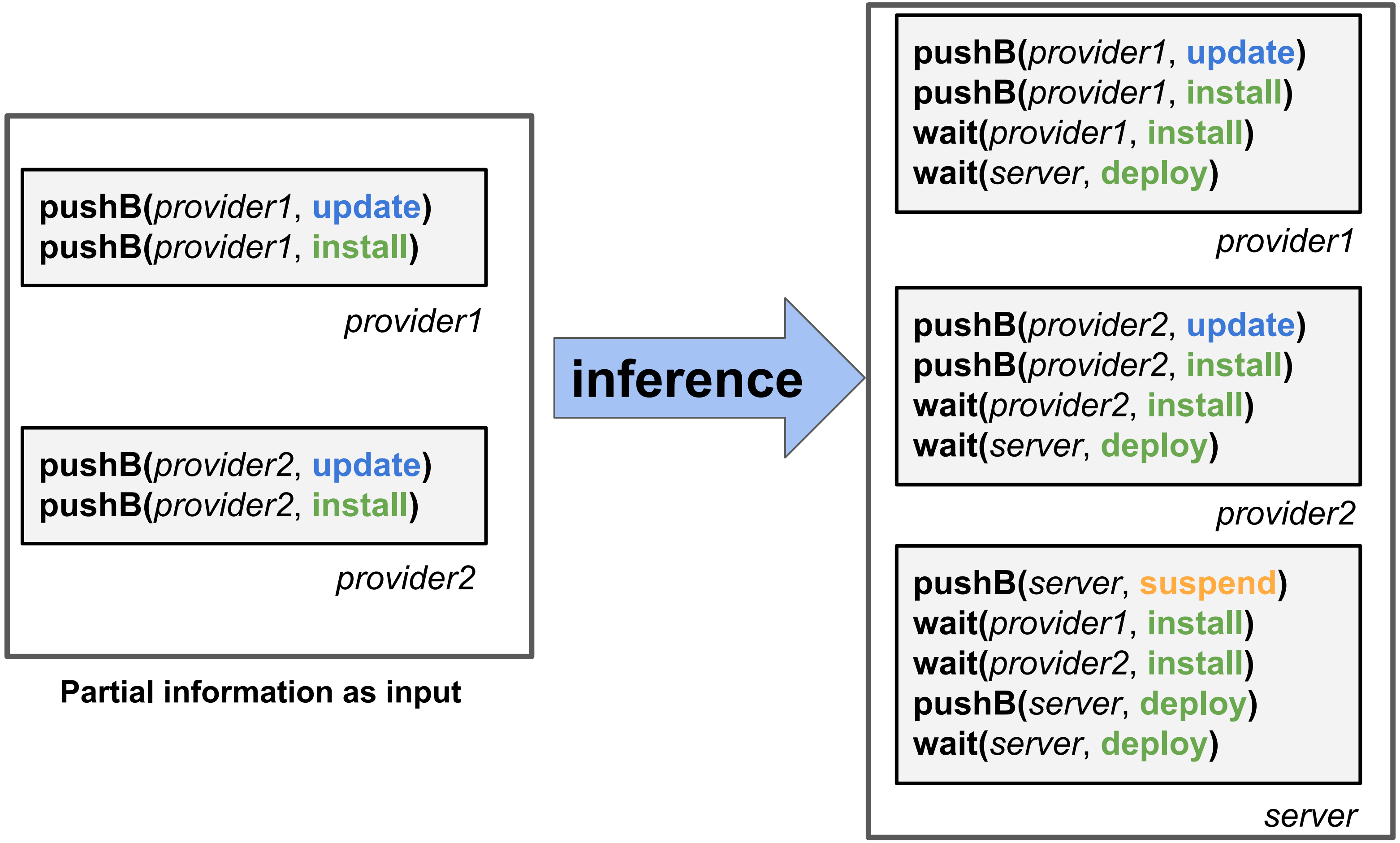
Concerto-D: State and changes in the component

non-blocking	**pushB(*provider*, update)**
non-blocking	**pushB(*provider*, install)**
blocking (syncro)	**wait(*provider*, install)**





Decentralized configuration plan



```
pushB(provider1, update)
pushB(provider1, install)
```

provider1

```
pushB(provider2, update)
pushB(provider2, install)
```

provider2

Partial information as input

```
pushB(provider1, update)
pushB(provider1, install)
wait(provider1, install)
wait(server, deploy)
```

provider1

```
pushB(provider2, update)
pushB(provider2, install)
wait(provider2, install)
wait(server, deploy)
```

provider2

```
pushB(server, suspend)
wait(provider1, install)
wait(provider2, install)
pushB(server, deploy)
wait(server, deploy)
```

server

Full reconfiguration plan as output

Decentralized planning of reconfiguration plans

For each component:

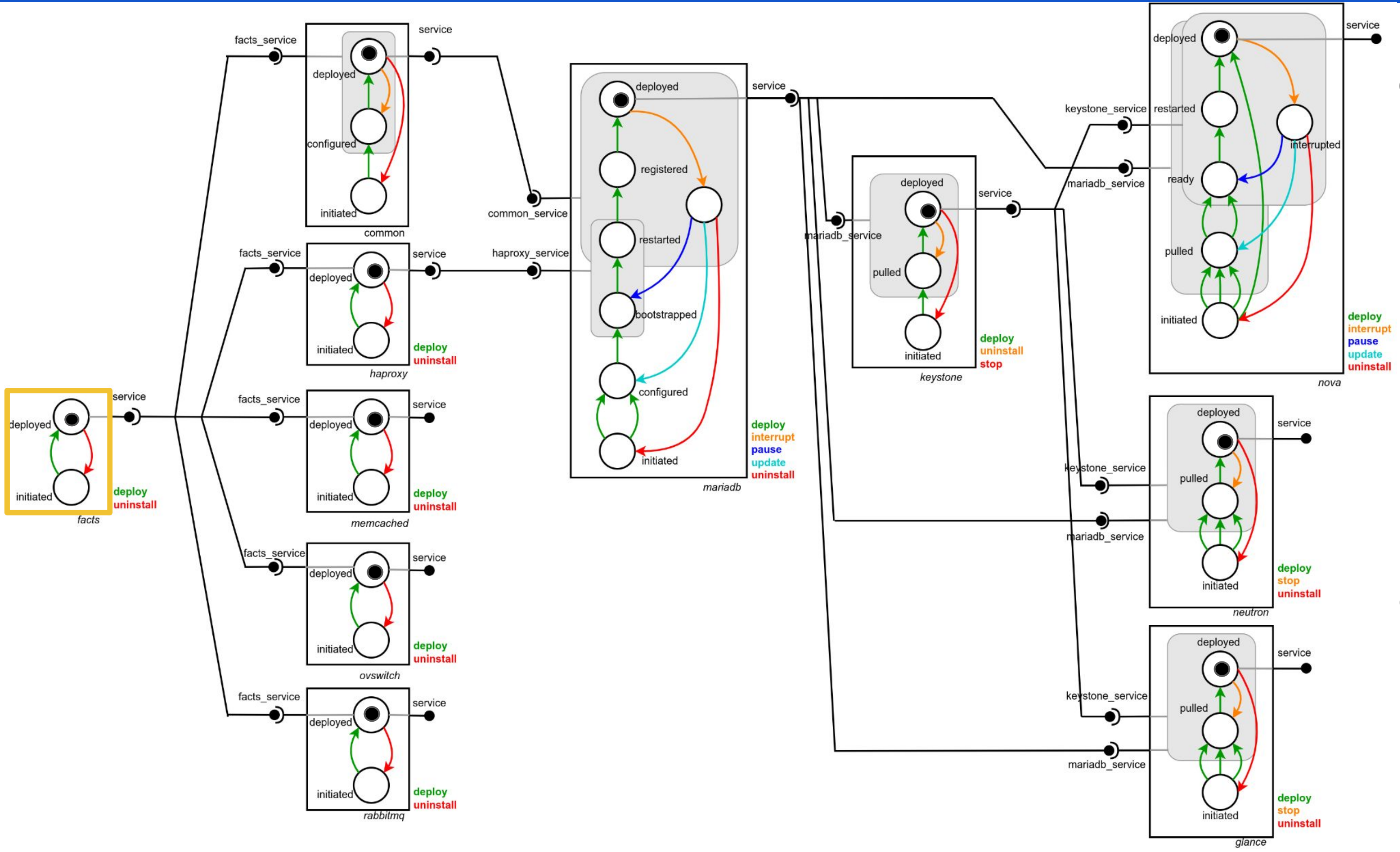
Inputs:
- Local decision of the target configuration **(WP2)**
- Set of possible reconfiguration instructions
- Partial view of the current configuration (state of the system)

Output:
- Reconfiguration plan (or program) to reach the targeted configuration

Intuition of the solution:

- **Sharing protocol** with message passing (impacted port) (rumor-spreading inspired)
  - **Local decision** (MiniZinc's automata)
    - **Inputs**: Current configuration + Input messages + Reconfiguration instructions
    - **Outputs**: Set of behaviors + Output messages
  - **Local planning**
    - **Inputs**: Set of behaviors + Output messages
    - **Output**: Reconfiguration plan

Example of stratified assembly and reconfiguration



11 components, all **deployed:**

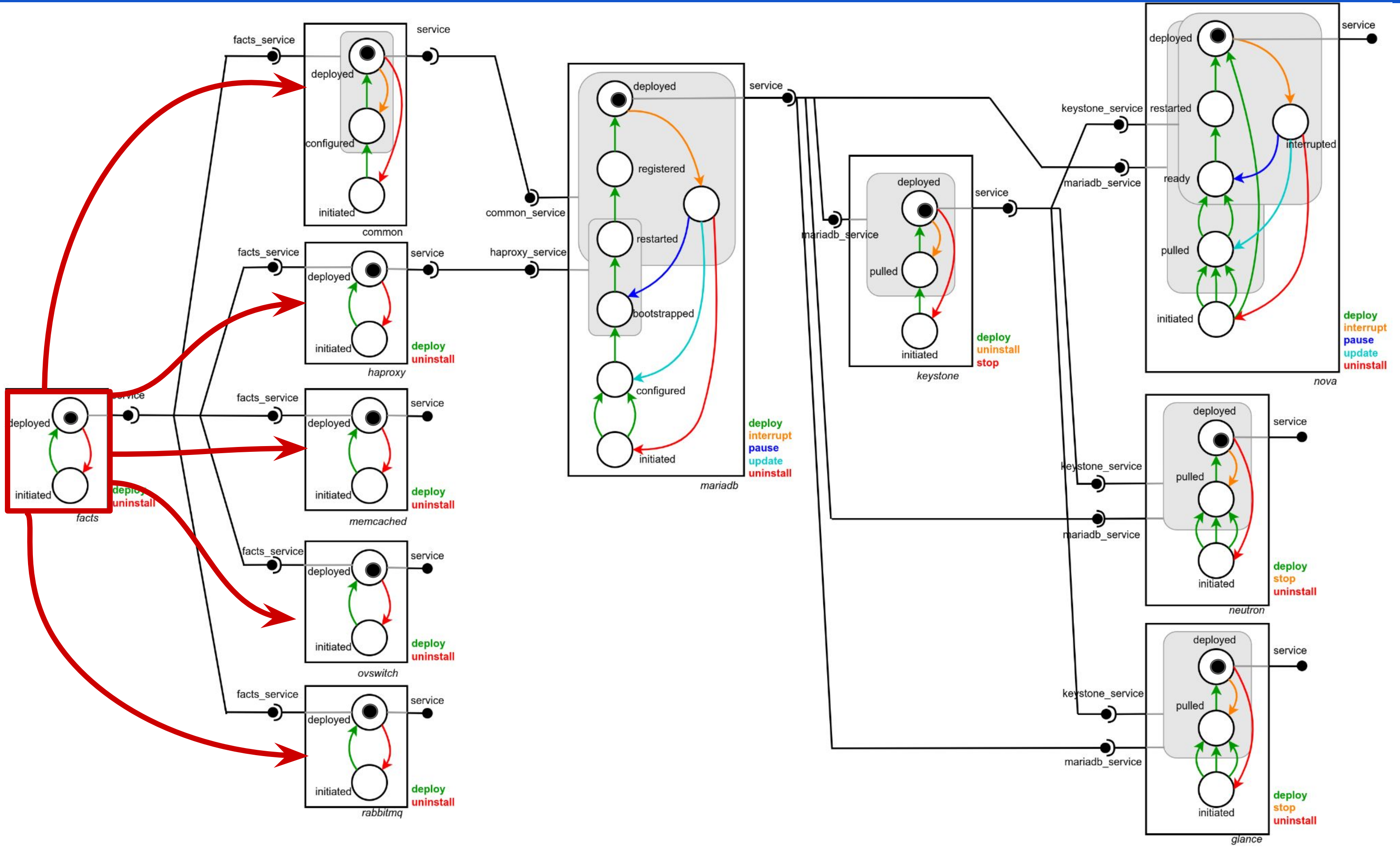
- *facts*
- *common*
- *haproxy*
- *memcached*
- *ovswitch*
- *rabbitmq*
- *mariadb*
- *keystone*
- *nova*
- *neutron*
- *glance*

Goal: reboot *facts*

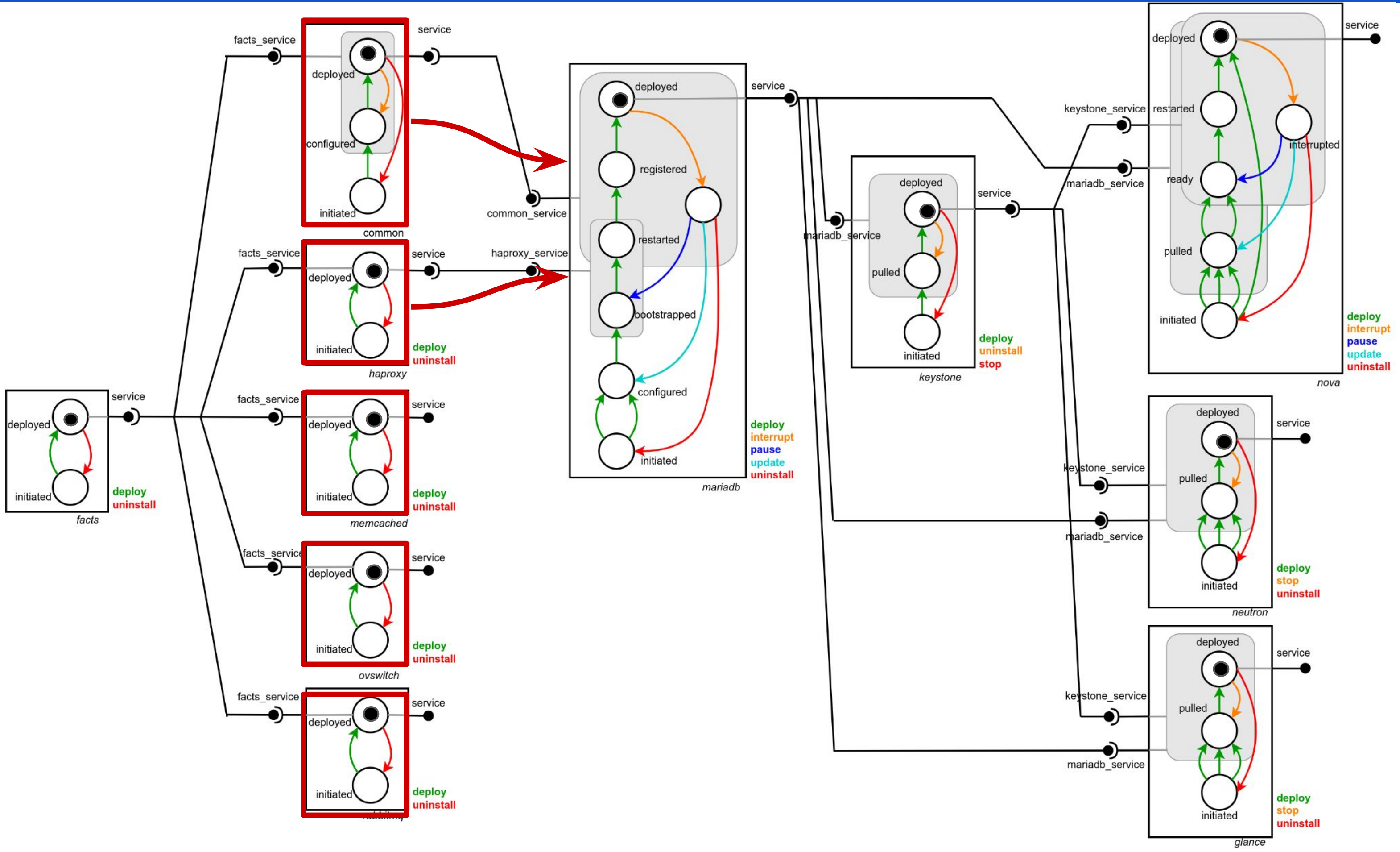
facts

pushB(facts, uninstall)
pushB(facts, deploy)

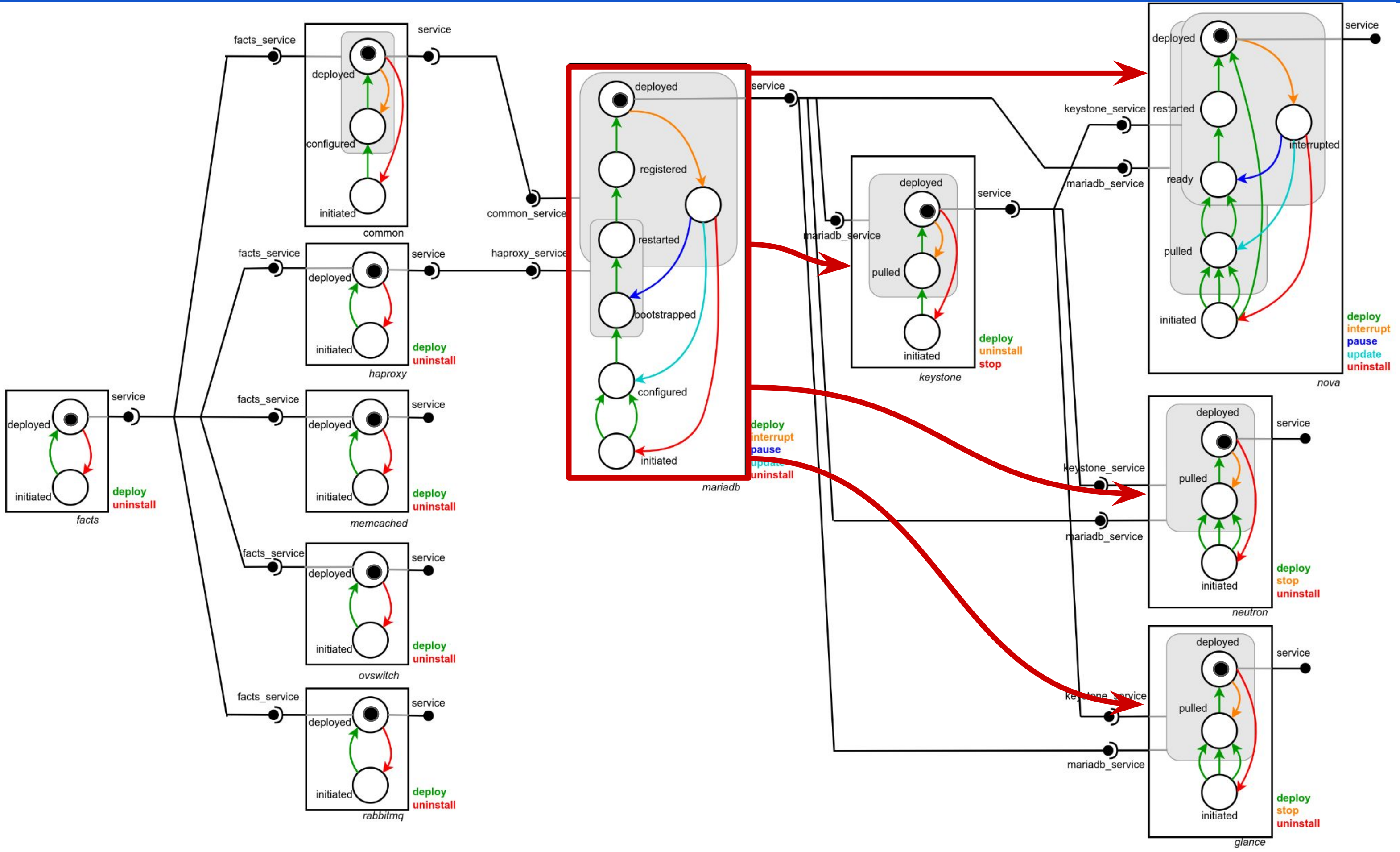
Information sharing protocol - Step I: Propose



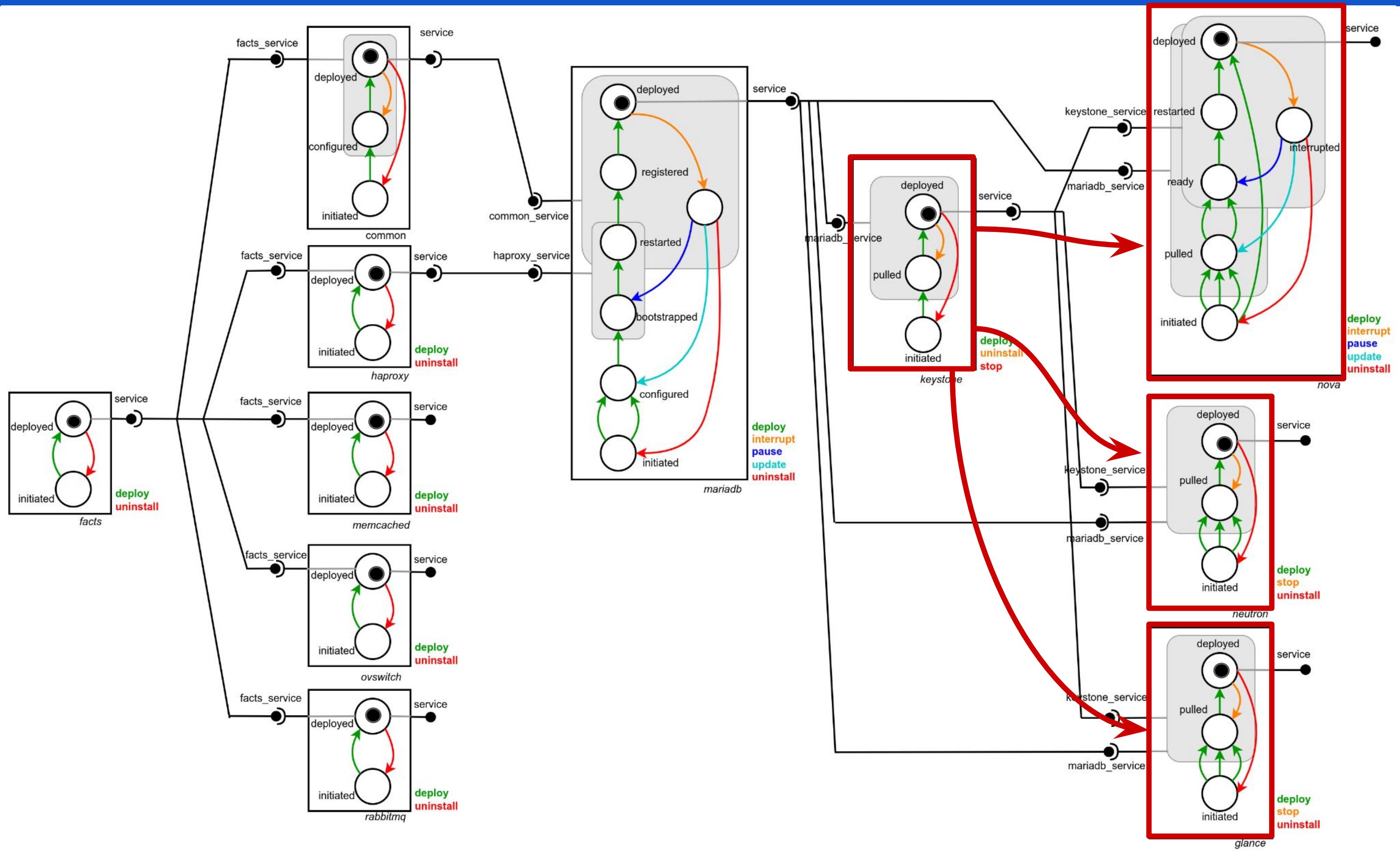
Information sharing protocol - Step I: Propose



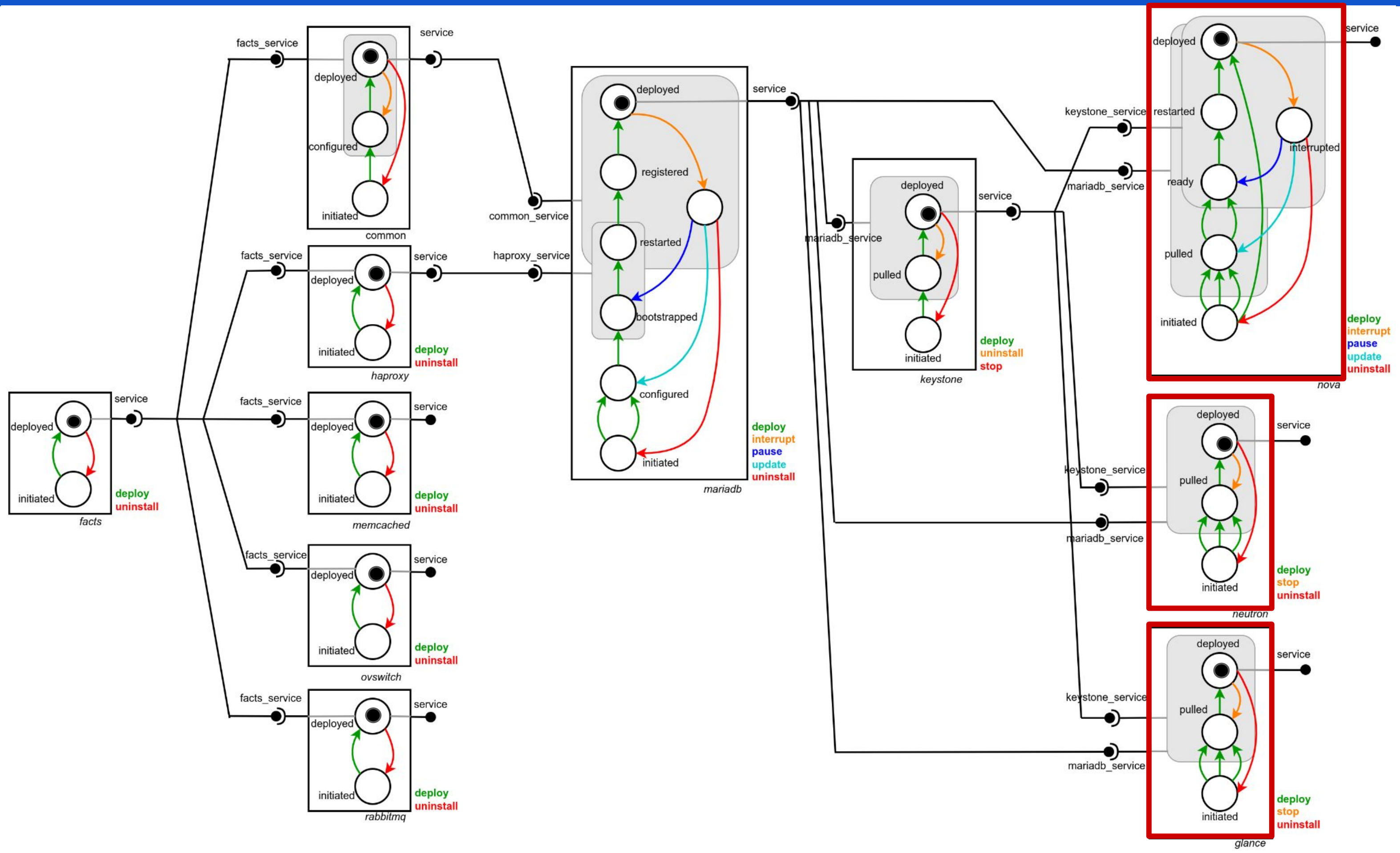
Information sharing protocol - Step I: Propose



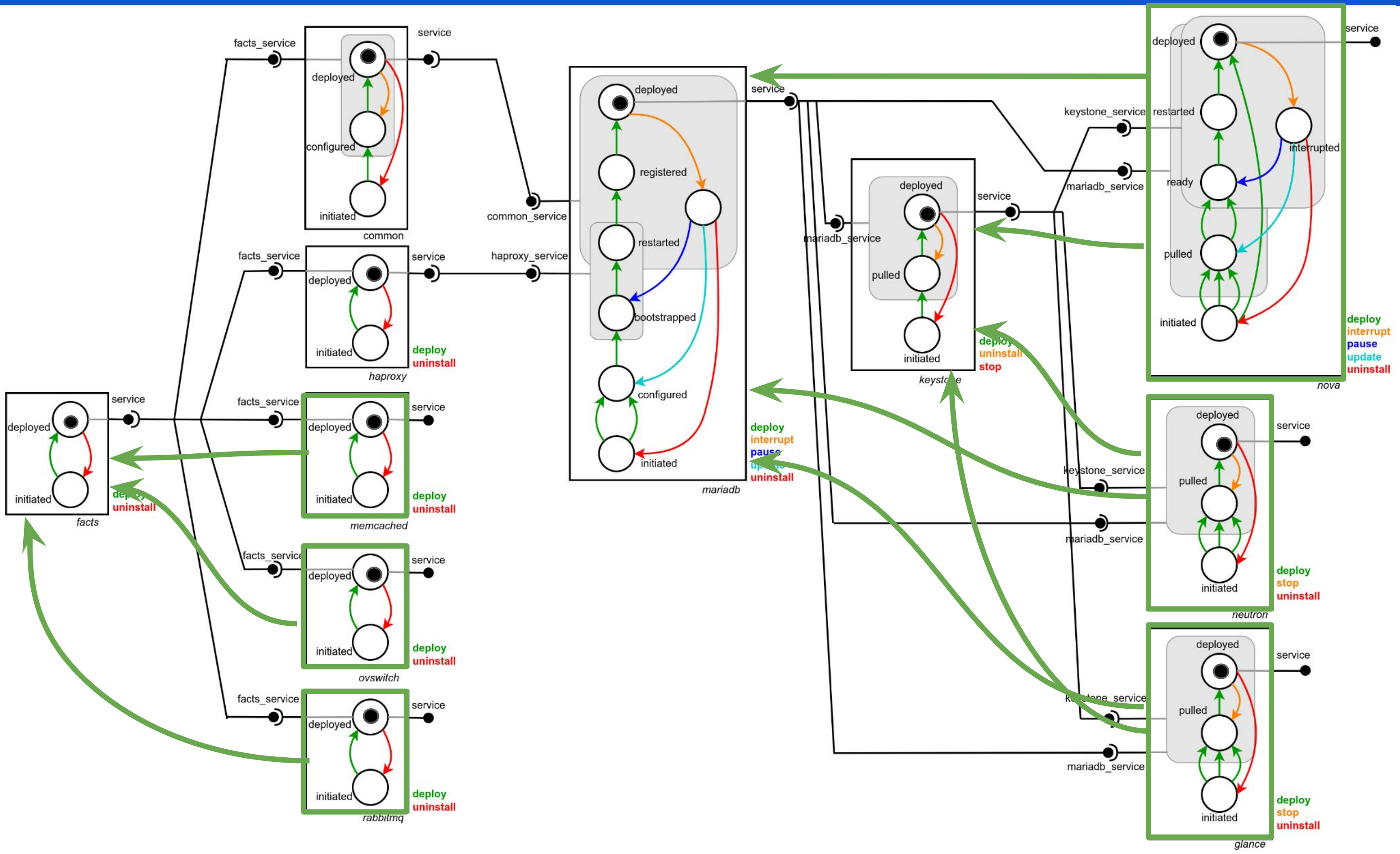
Information sharing protocol - Step I: Propose



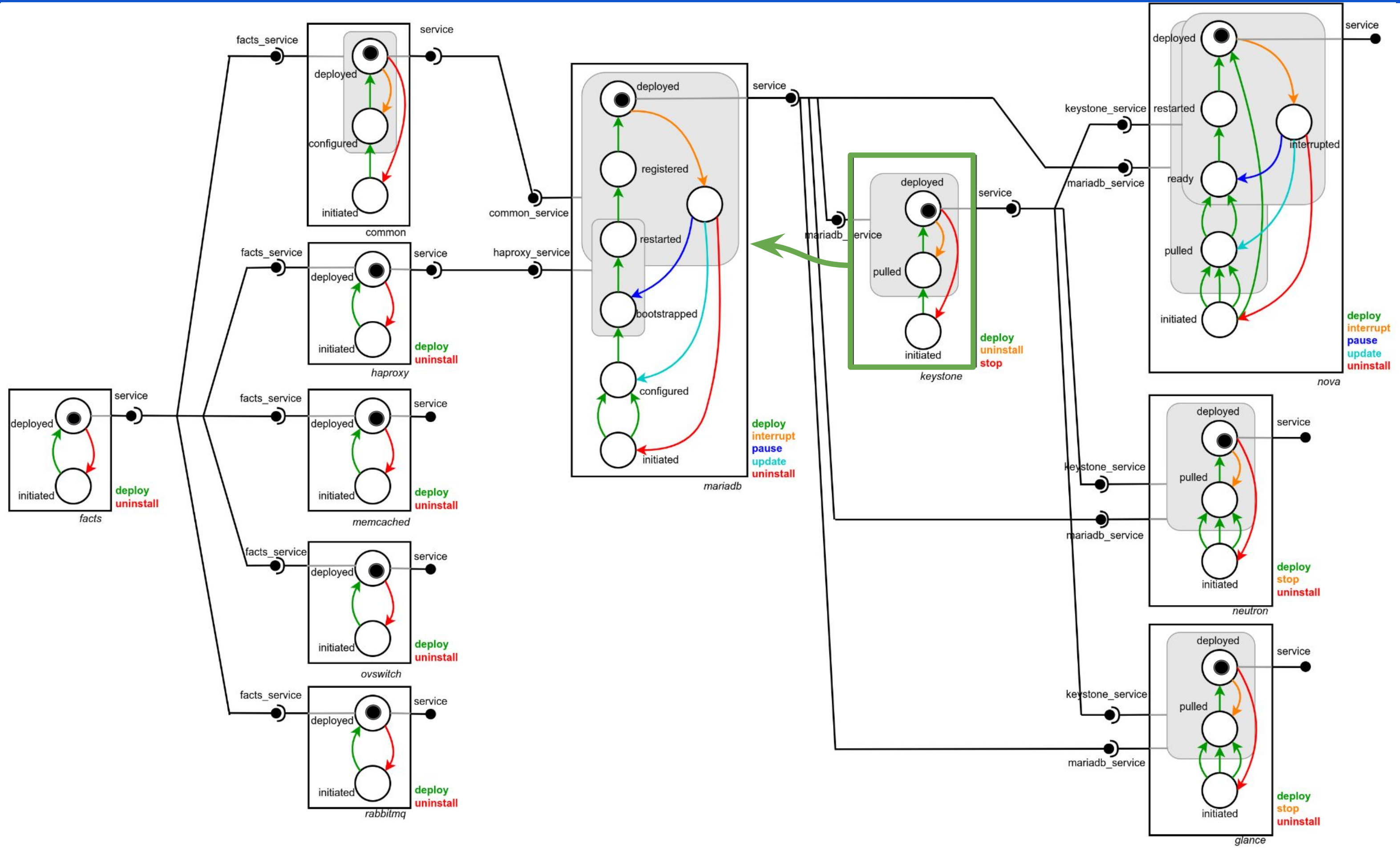
Information sharing protocol - Step I: Propose



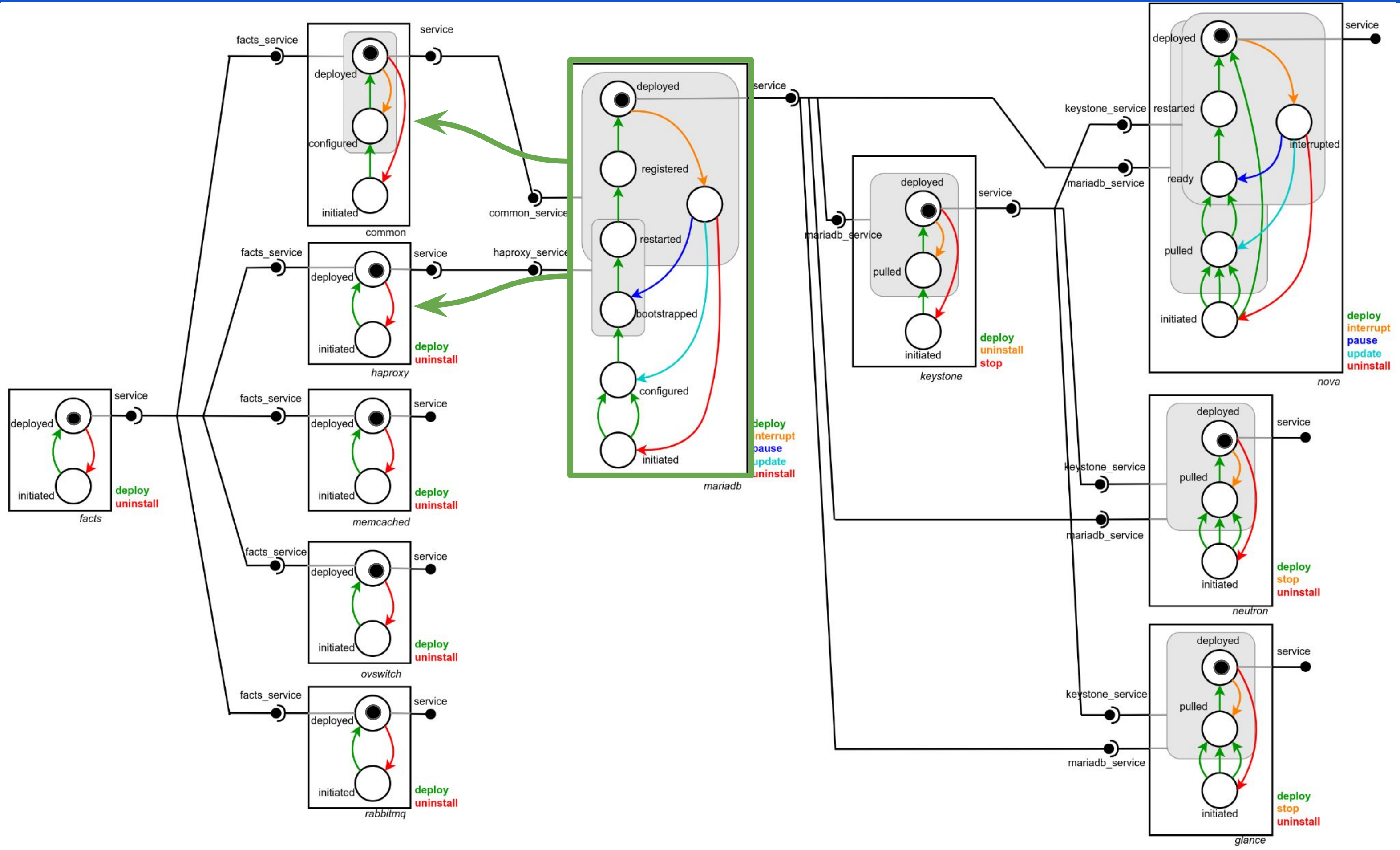
Information sharing protocol - Step II: Send ack



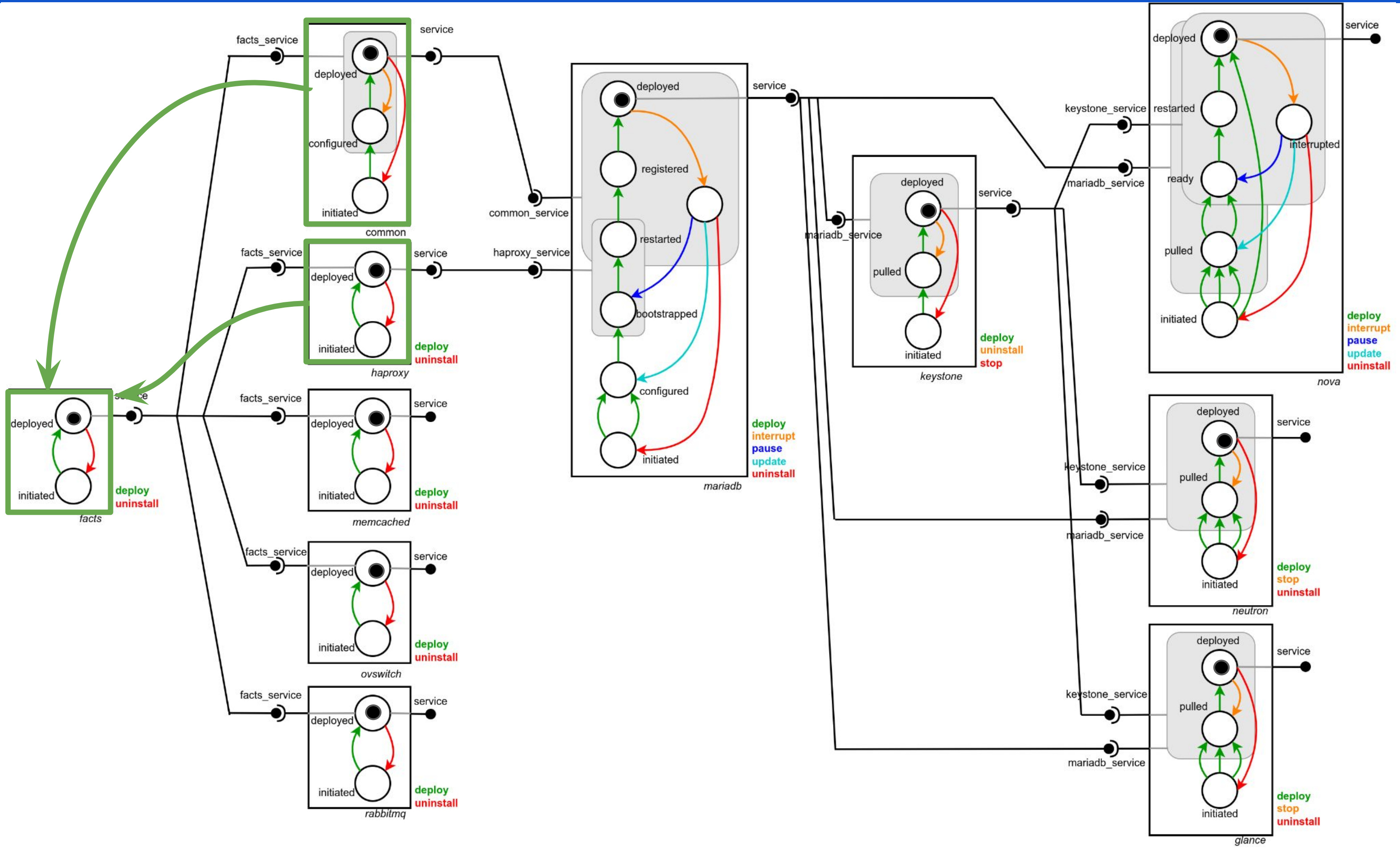
Information sharing protocol - Step II: Send ack



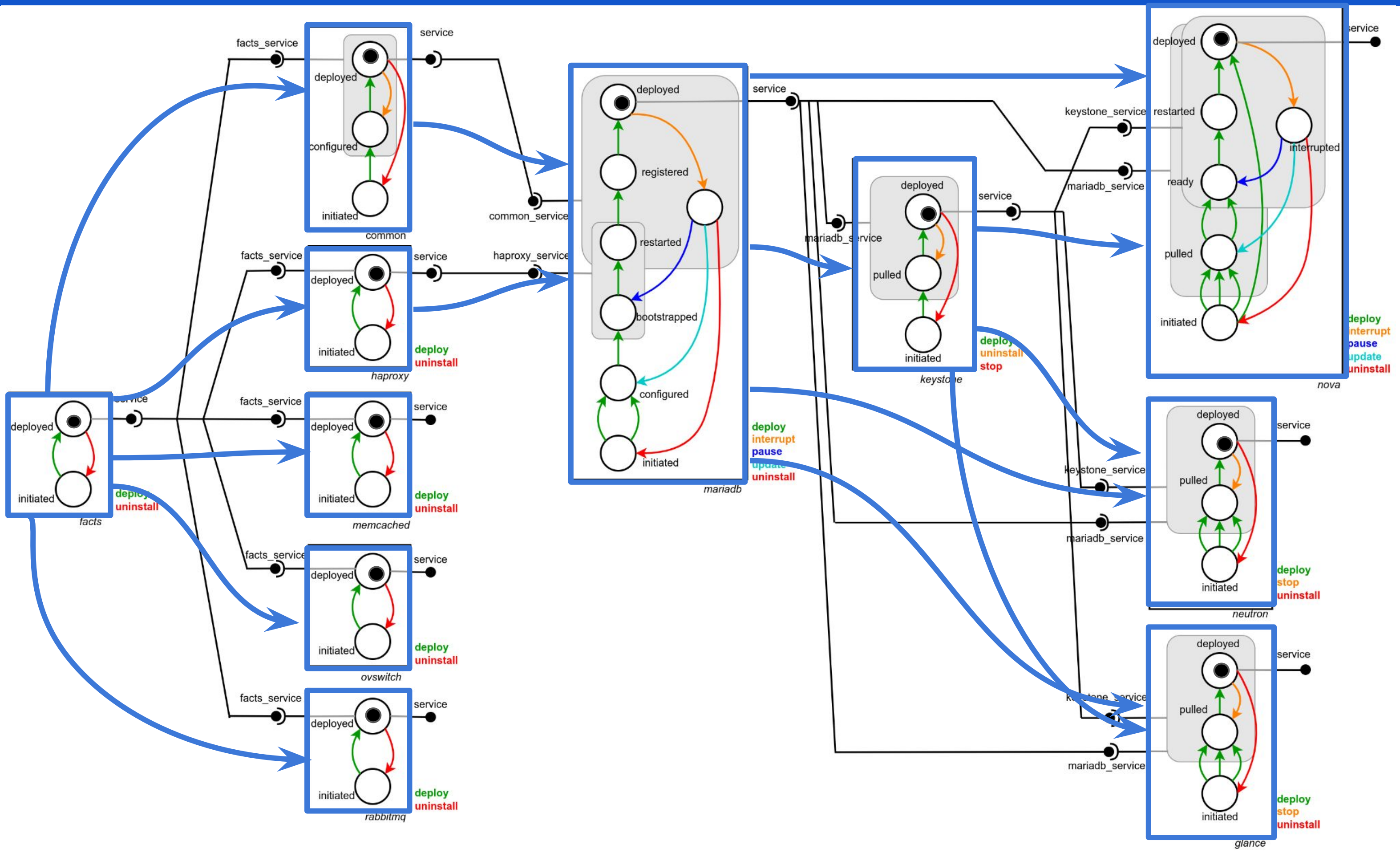
Information sharing protocol - Step II: Send ack



Information sharing protocol - Step II: Send ack



Information sharing protocol - Step III: Global ack from root

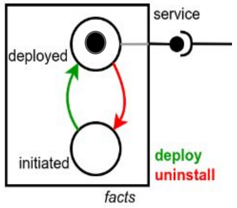


Local decision

Example 1:

facts

pushB(*facts*, **uninstall**)
pushB(*facts*, **deploy**)



Messages:

-

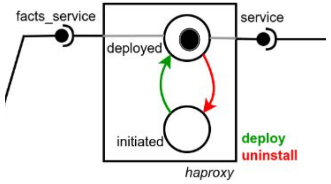
Local decision:
pushB(*facts*, **uninstall**)
pushB(*facts*, **deploy**)

Messages:

- (*facts*, *service*, **disabled - enabled, uninstall**)

Example 2:

haproxy



Messages:

- (*service*, **disabled - enabled, uninstall**)

Local decision:
pushB(*haproxy*, **uninstall**)
pushB(*haproxy*, **deploy**)

Messages:

- (*haproxy*, *service*, **disabled - enabled, uninstall**)

Local planning

Example 1:

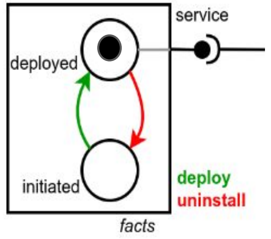
Local decision:
pushB(*facts*, **uninstall**)
pushB(*facts*, **deploy**)

Messages:

-

facts

pushB(*facts*, **uninstall**)
pushB(*facts*, **deploy**)



Example 2:

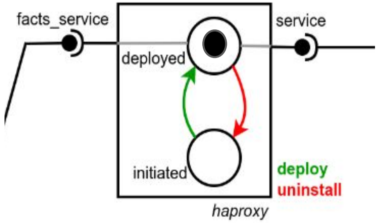
Local decision:
pushB(*haproxy*, **uninstall**)
pushB(*haproxy*, **deploy**)

Messages:

- (*service*, **disabled - enabled, uninstall**)

haproxy

pushB(*haproxy*, **uninstall**)
wait(facts, **uninstall**)
pushB(*haproxy*, **deploy**)



Several strategy: Brute Force; CP-based

Pseudo-code for sharing protocol

```
function decentralized_plan(comp, targeted_state, roots):
    while(true):
        messages = get_messages(comp)
        If ((messages.empty and comp.in(roots)) or message.size > 0):
            bhvs, ports = local_decision(comp, messages, targeted_state)
            If (not ports.empty):
                msgs = send_messages(comp.neighbors, ports)
                sent_msgs = sent_msgs ++ msgs

        If (sent_msgs.allAcked):
            send_ack(sent_msgs.sources)

        If (sent_msgs.allAcked and comp.in(roots)):
            bcast_ack(comp)

        If (roots.allAcked):
            return local_plan(bhvs, messages)
```

Conclusion

- Lack of decentralized planning for distributed system reconfigurations
- Propose a decentralized solution based on a sharing protocol
- Information sharing protocol
  - Local decision
  - Local planning

Questions?