Efficient Reconfigurations with Programmable Life Cycles: Contributions to Safety, Declarativity, and Decentralization

Hélène Coullon
IMT Atlantique, Inria, LS2N

Habilitation à Diriger des Recherches
Le 23/04/2025

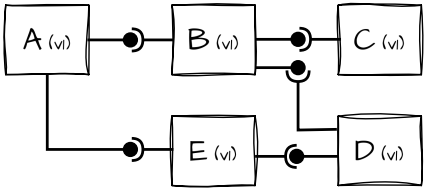
Rapporteurs : Laurence Duchien, Etienne Rivière, Eric Rutten
Examinateurs : Fabienne Boyer, Dalila Tamzalit, Thomas Ledoux

Past

- **[2004-2007]** ENSI Bourges (INSA Centre Val de Loire)
  - System administration and Security
- **[2007-2009]** Software engineer at Dassault Systèmes
  - Internal tools for CI/CD
- **[2009-2011]** Research engineer at LIFO, Orléans
  - Parallel codes for hydrogeology
- **[2011-2014]** PhD Student, CIFRE Géo-Hyd, LIFO, Orléans
  - Domain specific language for numerical methods
  - Parallelism and High Performance Computing
  - Under the supervision of Sébastien Limet
- **[2014-2016]** Postdoc, Avalon, INRIA, LIP, Lyon
  - Using Component-Based Software Engineering for HPC
  - Under the supervision of Christian Perez

What this defense talks about?

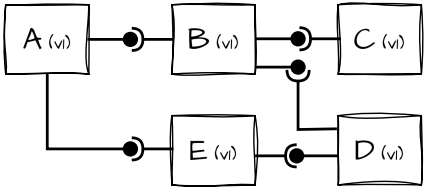
Service-Oriented (SO) distributed systems



- Decoupled components
- Use/provide interfaces for composition

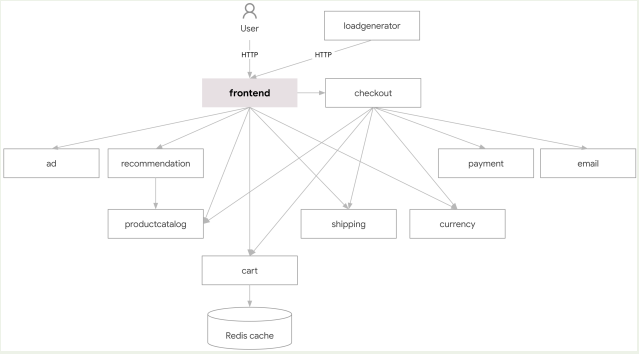
What this defense talks about?

Service-Oriented (SO) distributed systems



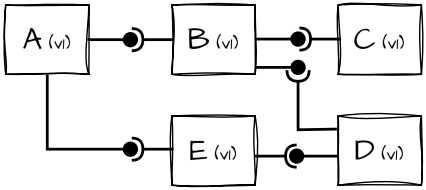
- Decoupled components
- Use/provide interfaces for composition

Online boutique

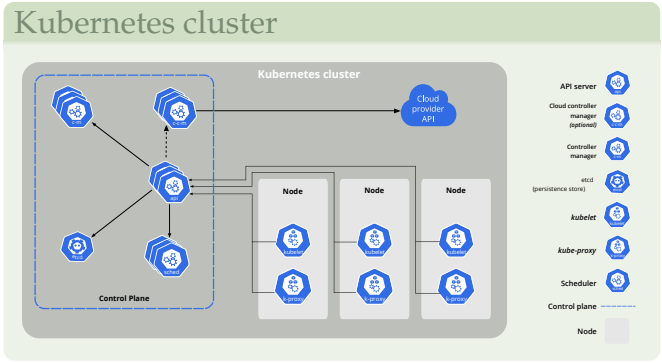


What this defense talks about?

Service-Oriented (SO) distributed systems

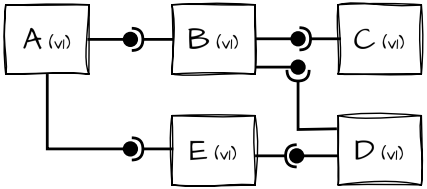


- Decoupled components
- Use/provide interfaces for composition

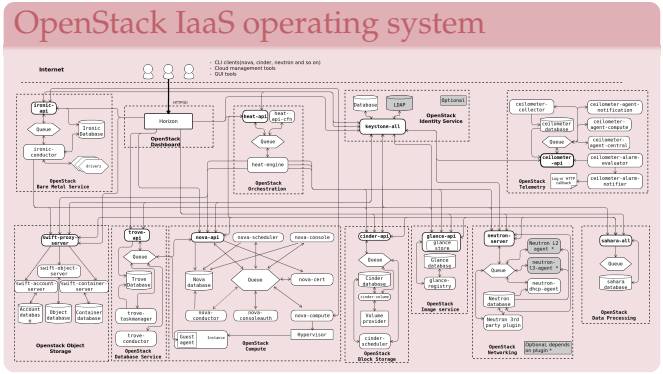


What this defense talks about?

Service-Oriented (SO) distributed systems



- Decoupled components
- Use/provide interfaces for composition

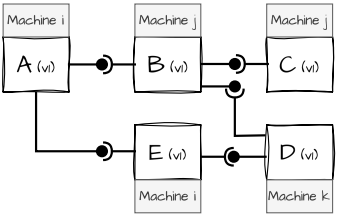


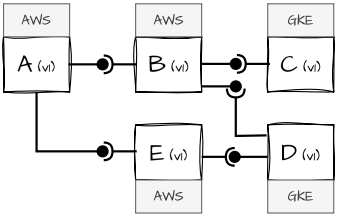
Designing and writing such SO systems: CBSE, SOA, micro-services architectures, etc.

What this defense talks about?

Deploy and operate

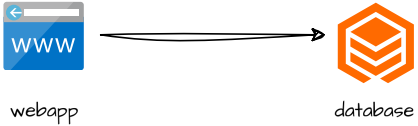
Distributed systems live on distant machines or platforms

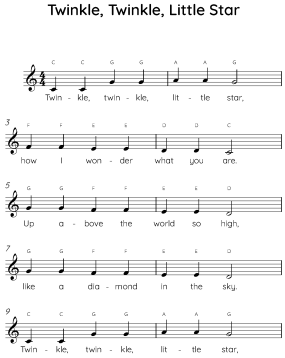




Why is it complex?

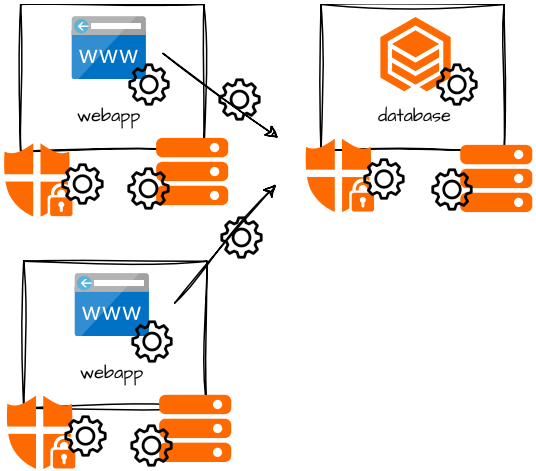
Deploying is not only installing packages or images





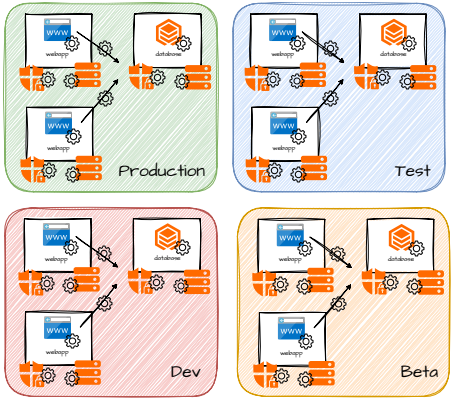
WHY IS IT COMPLEX?

DEPLOYING IS NOT ONLY INSTALLING PACKAGES OR IMAGES

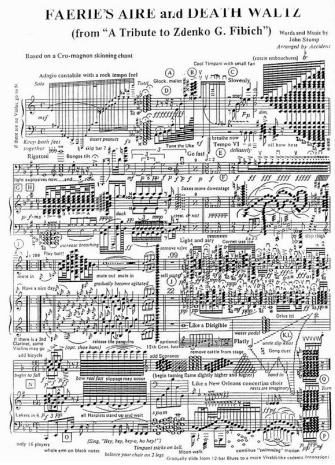


Why is it complex?

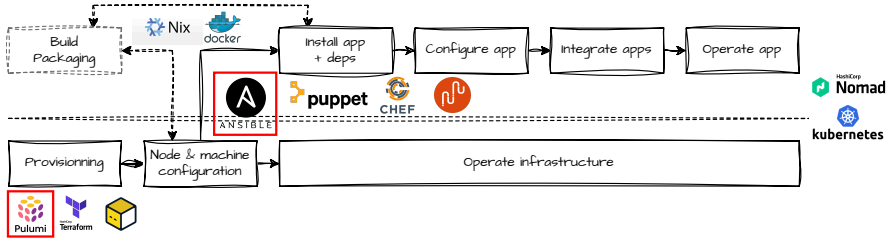
Deploying is not only installing packages or images



+ operate



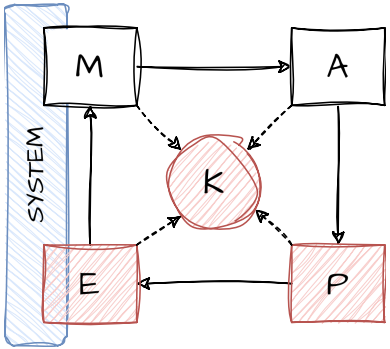
How to deploy and operate automatically?



Examples of tasks (system calls, tool calls)

- `apt install` a Debian package
- write a `x.conf` file somewhere
- mount a volume inside a container

How to deploy and operate automatically?

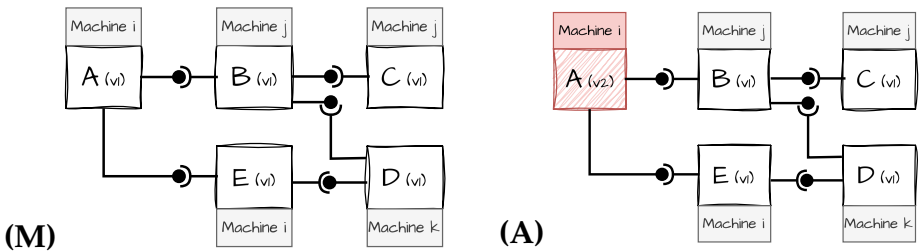


- M = Monitor
- A = Analyze
- P = Plan
  - (Declarativity)
- E = Execute
- K = Knowledge

***Reconfiguration**: focus on P, E and associated K*

How to deploy and operate automatically?

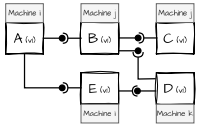
Update



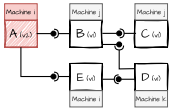
What is the reconfiguration plan?

How to deploy and operate automatically?

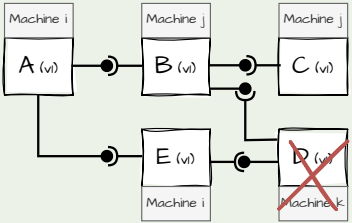
Update



(M)



(A)

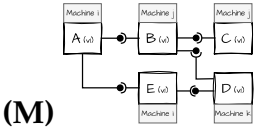


(P & E)

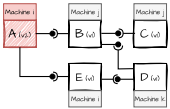
delete(D)

How to deploy and operate automatically?

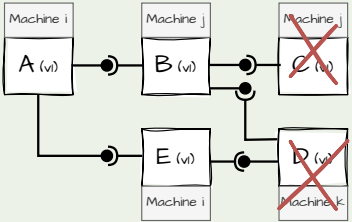
Update



(M)



(A)

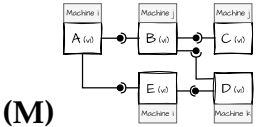


(P & E)

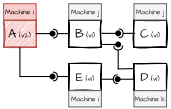
delete(D); delete(C)

How to deploy and operate automatically?

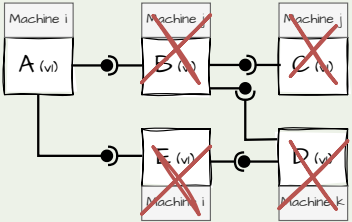
Update



(M)



(A)

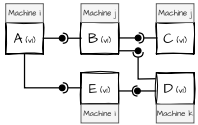


(P & E)

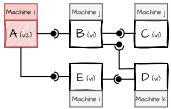
delete(D); delete(C); delete(E); delete(B)

How to deploy and operate automatically?

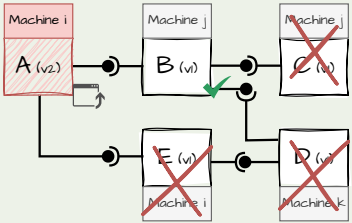
Update



(M)



(A)

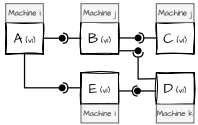


(P & E)

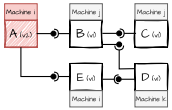
delete(D); delete(C); delete(E); delete(B); update(A); start(B)

How to deploy and operate automatically?

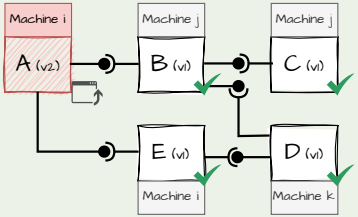
Update



(M)



(A)

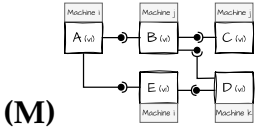


(P & E)

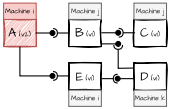
delete(D); delete(C); delete(E); delete(B); update(A); start(B); start(E), start(C); start(D)

How to deploy and operate automatically?

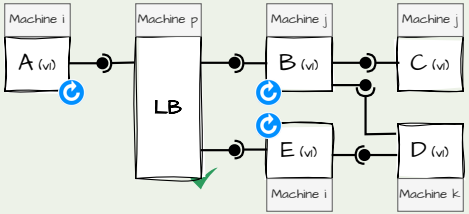
More than one plan is possible



(M)



(A)

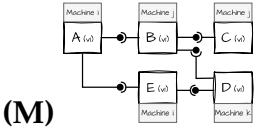


(P & E)

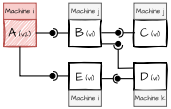
start(LB); restart(B); restart(A); restart(E)

How to deploy and operate automatically?

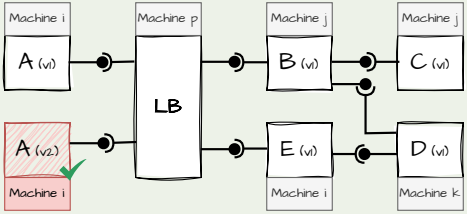
More than one plan is possible



(M)



(A)

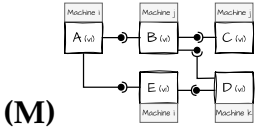


(P & E)

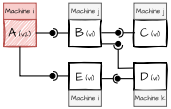
start(LB); restart(B); restart(A(v1)); restart(E); start(A(v2))

How to deploy and operate automatically?

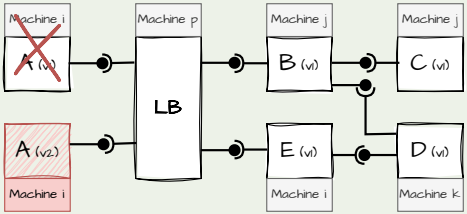
More than one plan is possible



(M)



(A)



(P & E)

start(LB); restart(B); restart(A(v1)); restart(E); start(A(v2)); delete(A(v1))

What is this defense really about?

Since 2016

I am interested in **leveraging the life cycles** of software entities in a distributed system to **improve their automatic reconfigurability**

- Modeling the life cycles
- Coordinating the life cycles of a distributed system

4 scientific challenges have been studied

1. Improve efficiency of reconfigurations
2. Enable declarativity (P) while keeping efficiency
3. Enable decentralized reconfigurations
4. Offer safer reconfigurations

Outline

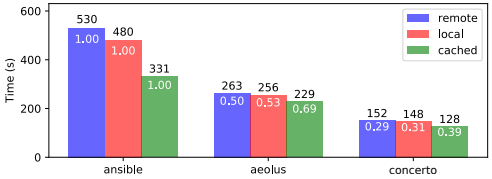
EFFICIENCY

- Why efficiency?
- Concerto
- Instrumentalists

WHY EFFICIENCY?

Minimal OpenStack deployment

- 11 services



Artifacts Grid'5000

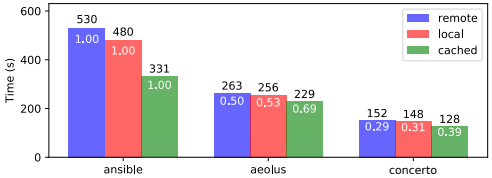
Up to 70% gain compared to Ansible

During release periods OpenStack can be deployed hundreds of times a day

WHY EFFICIENCY?

Minimal OpenStack deployment

- 11 services



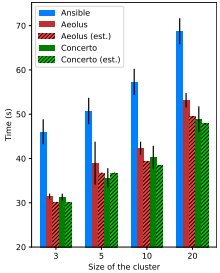
Artifacts Grid'5000

Up to 70% gain compared to Ansible

During release periods OpenStack can be deployed hundreds of times a day

Reconf. multi-site OpenStack

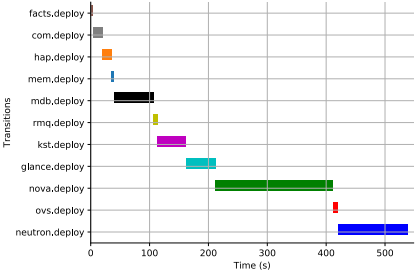
- OpenStack Summit 2018
- Galera cluster of DB

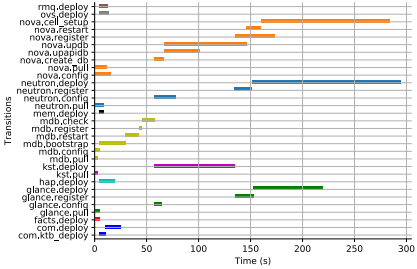


Artifacts Grid'5000

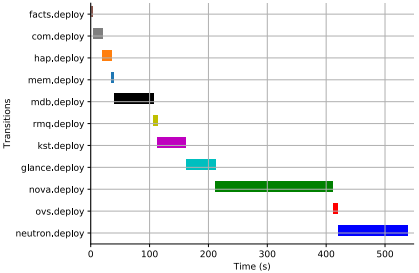
Up to 40% gain compared to Ansible

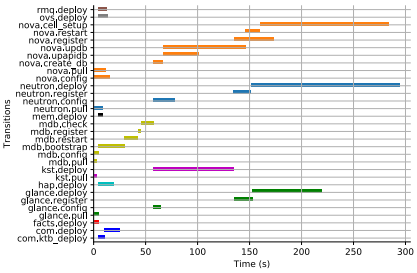
PARALLELISM IN RECONFIGURATION





Parallelism in reconfiguration





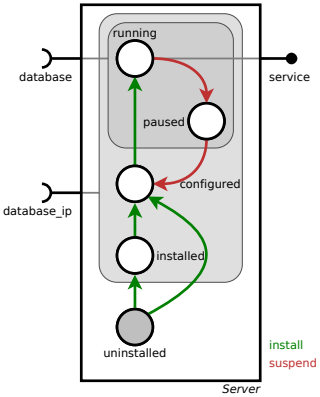
Reconfigurations are DAGs (Directed Acyclic Graphs) of (system) actions

However,

- Life cycle abstraction is required for developers, DevOps, and (P)
- Flexible enough abstractions to not lose opportunities for parallelism

CONCERTO - CONTROL COMPONENTS

PROGRAMMABLE LIFE CYCLE WITH FINE-GRAINED DEPENDENCIES



Written by the component developers

Internal net: Models the life cycle of a component

- places = milestones
- transitions = concrete actions to perform

Interfaces

- ports (CBSE)
  - use ports = requirements
  - provide ports = provisions
  - during execution: active/inactive
- behaviors (subset of transitions)
  - actions on the life cycle

CONCERTO - RECONFIGURATION LANGUAGE

INTERACT WITH THE COMPONENTS' LIFE CYCLES

Used by DevOps, or a declarative tool

Create assemblies and make them evolve at runtime (CBSE)

- `add/delete` a component instance
- `connect/disconnect` two component instances

CONCERTO - RECONFIGURATION LANGUAGE

INTERACT WITH THE COMPONENTS' LIFE CYCLES

Used by DevOps, or a declarative tool

Create assemblies and make them evolve at runtime (CBSE)

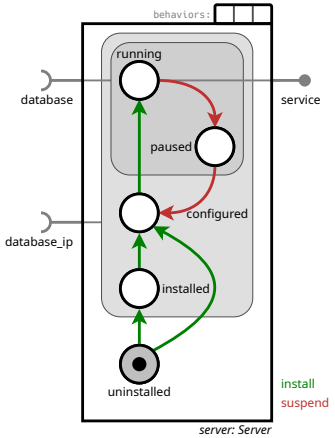
- `add/delete` a component instance
- `connect/disconnect` two component instances

Interact with the lifecycle of components

- `push` a behavior to the behavior queue on a component instance
- `wait` for a given component instance or wait for all components

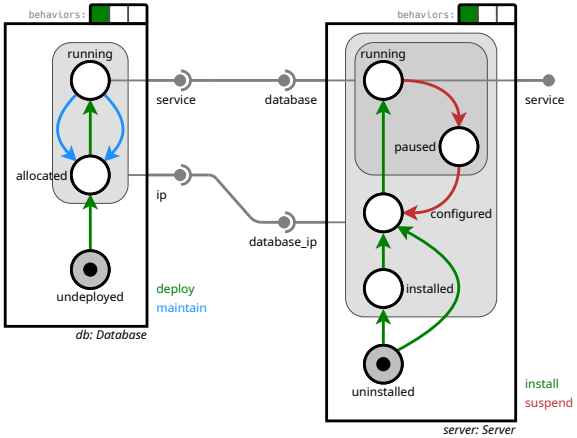
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



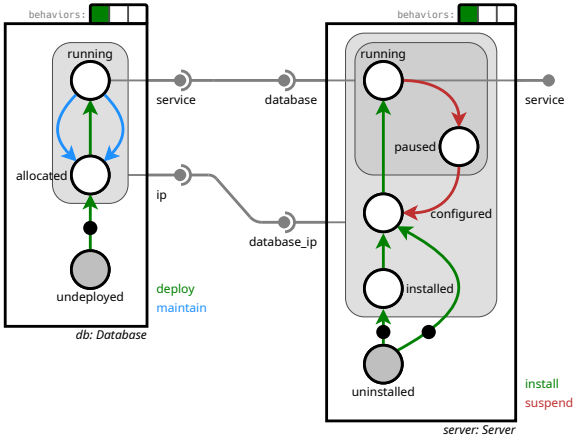
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



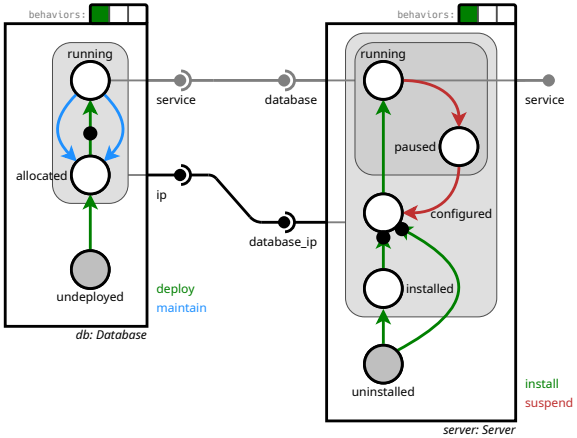
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



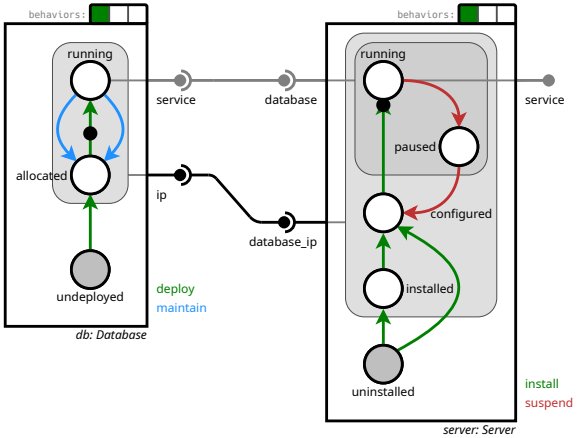
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



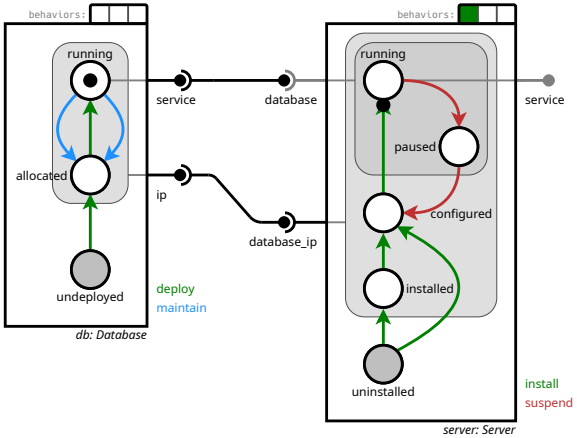
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



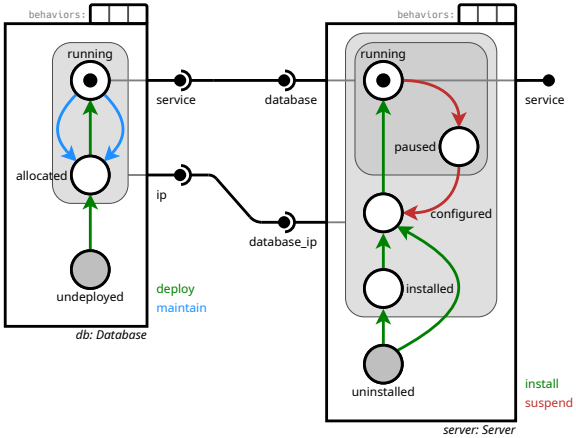
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



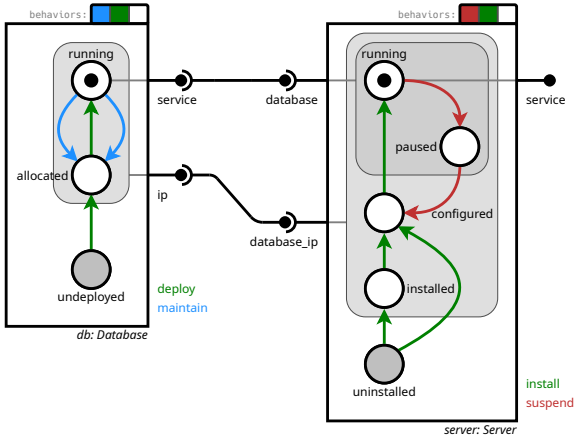
RECONFIGURATION EXAMPLE - DEPLOYMENT

add(server: Server)
add(db: Database)
con(server.database_ip,db.ip)
con(server.database,db.service)
pushB(server,install)
pushB(db,deploy)
wait(server)



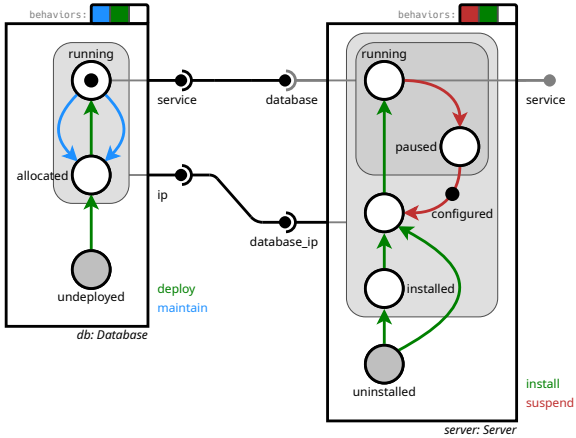
RECONFIGURATION EXAMPLE - MAINTENANCE

pushB(db,maintain)
pushB(db,deploy)
pushB(server,suspend)
pushB(server,install)
wait(server)



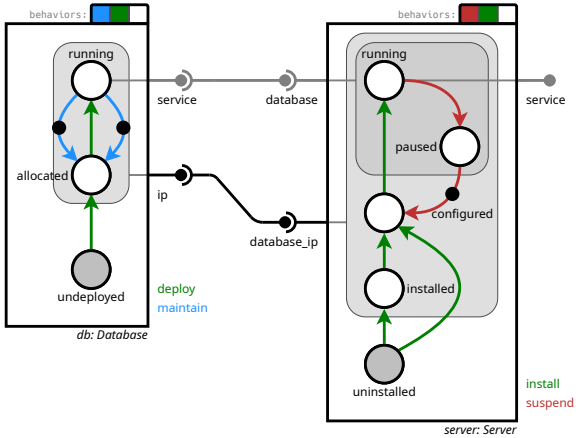
RECONFIGURATION EXAMPLE - MAINTENANCE

pushB(db,maintain)
pushB(db,deploy)
pushB(server,suspend)
pushB(server,install)
wait(server)

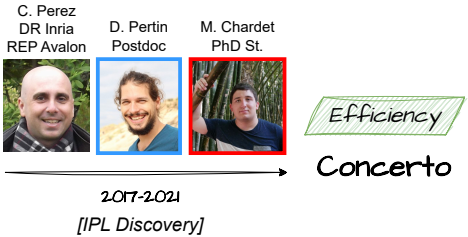


RECONFIGURATION EXAMPLE - MAINTENANCE

pushB(db,maintain)
pushB(db,deploy)
pushB(server,suspend)
pushB(server,install)
wait(server)



CONCERTO - INSTRUMENTALISTS



- Maverick Chardet, Hélène Coullon, Simon Robillard. *Toward Safe and Efficient Reconfiguration with Concerto.* In SCP, 2021
- Maverick Chardet, Hélène Coullon, Christian Perez. *Predictable Efficiency for Reconfiguration of Service-Oriented Systems with Concerto.* In CCGrid 2020, Melbourne, Australia

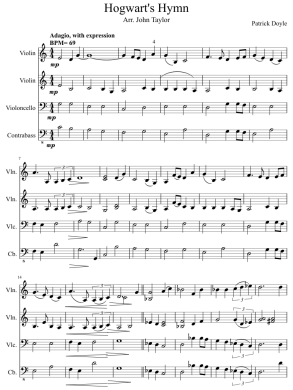
Declarativity

- Why declarativity?
- How to make Concerto declarative?
- Composers

Why declarativity?

I want a music that sounds magical

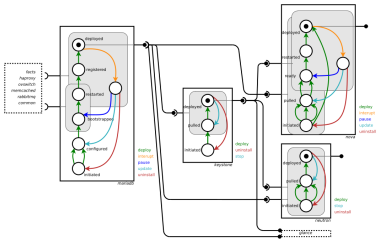
→



*We need a **composer**!*

WHY DECLARATIVITY?

I want to get a decentralized database



- *No need to write Concerto programs*
- *No errors due to parallelism*

```
pushB (m_sysbench, suspend)
pushB (m_mariadb, backup)
pushB (m_mariadb, uninstall)
con (m_docker, apt_docker,
     m_apt, apt_docker)
add (w_gconf, ClusterJoinConfig)
for i in [1,n]:
  add (w{i}_mariadb, MariaDBWorker)
  con (w{i}_mariadb, config,
       w_gconf, data)
  con (w{i}_mariadb, docker,
       w{i}_docker, docker)
  con (w{i}_mariadb, pip_libs,
       w{i}_piplibs, pip_libs)
  pushB (w{i}_mariadb, install)
wait(m_mariadb)
dcon (m_mariadb, config,
      m_sconf, data)
del (m_sconf)
add (m_gconf, ClusterInitConfig)
con (m_mariadb, config,
     m_gconf, data)
pushB (m_mariadb, install)
pushB (m_mariadb, restore_run)
pushB (m_sysbench, install)
for i in [1,n]:
  con (w{i}_mariadb, master,
       m_mariadb, mariadb)
```

*We need a **planner**!*

HOW TO MAKE CONCERTO DECLARATIVE?

PRINCIPLE

- **inputs:** current state of the system (M) + reconfiguration goals (A)
  - set of behaviors to execute on designated components
  - constraints on the final state of ports
- **output:** a Concerto reconfiguration program
  - `pushB` requests
  - `wait` commands
- About creations/deletions/(dis)connections
  - usual to handle topological changes before and after behaviors requests and synchronization (e.g., deployment)

Problem formulation

Find a valid (optimal) schedule of `pushB` and `wait` instructions

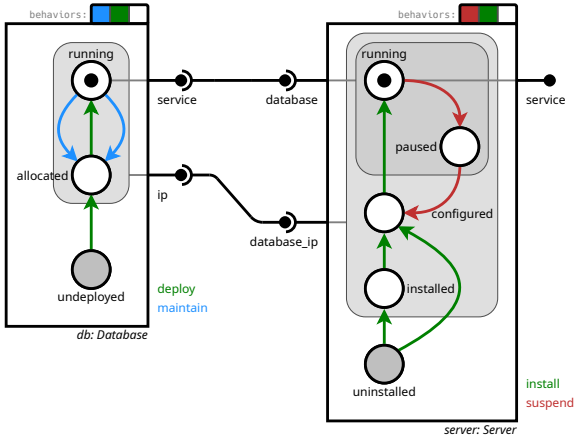
RECONFIGURATION EXAMPLE - MAINTENANCE

Goal

behaviors:
- ***component***: *db*
  behavior: *maintain*

components:
- ***forall***: *running*

pushB(db, maintain)
pushB(db, deploy)
pushB(server, suspend)
pushB(server, install)
wait(server)



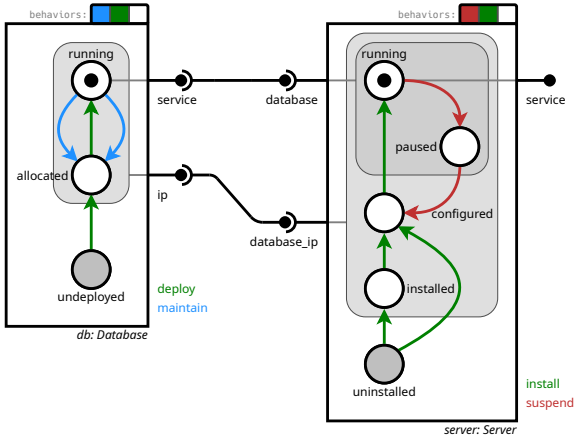
RECONFIGURATION EXAMPLE - MAINTENANCE

Goal

behaviors:
- ***component***: *db*
  behavior: *maintain*

components:
- ***forall***: *running*

pushB(db, maintain)
pushB(db, deploy)
pushB(server, suspend)
wait(db)
pushB(server, install)
wait(server)



RECONFIGURATION EXAMPLE - MAINTENANCE

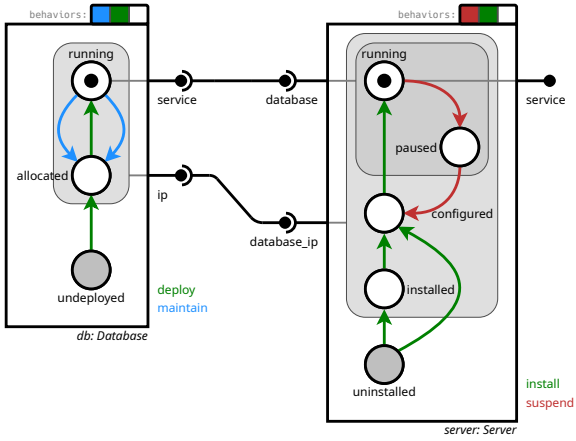
Goal

behaviors:
- ***component***: *db*
  behavior: *maintain*

components:
- ***forall***: *running*

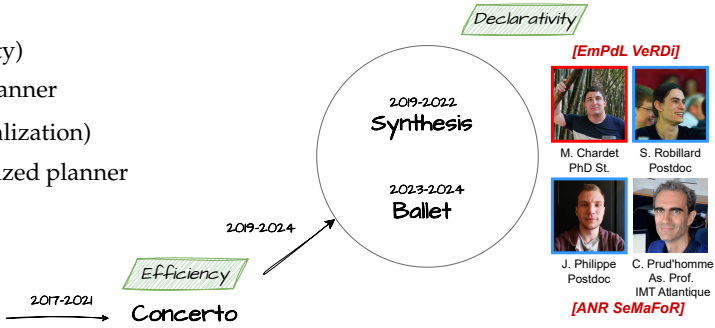
Needed to solve the problem

- Constraints on valid behaviors sequences
- Information on activation and deactivation of ports when applying behaviors



Composers

1. Synthesis (safety)
   - central planner
2. Ballet (decentralization)
   - decentralized planner



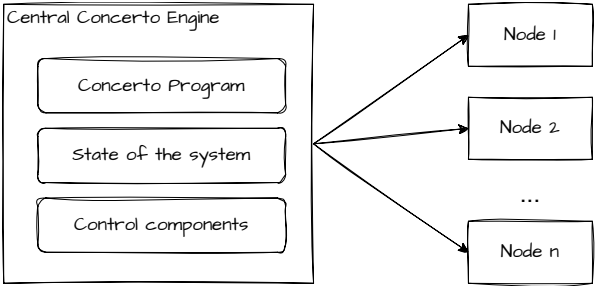
- Jolan Philippe, Antoine Omond, Hélène Coullon, Charles Prud'homme, Issam Rais. *Fast Choreography of Cross-DevOps Reconfiguration with Ballet: A Multi-Site OpenStack Case Study*. In IEEE SANER 2024, Finland
- Simon Robillard, Hélène Coullon. *SMT-Based Planning Synthesis for Distributed System Reconfigurations*. In FASE 2022, Munich, Germany

Decentralization

- Why decentralization?
- Ballet
- Dancers

Why decentralization?

Limitations of a central vision

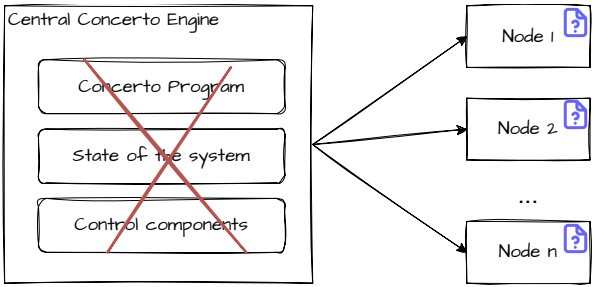


Concerto

- Central entity with the full knowledge
- Nodes simply receive the tasks of a transition
- No knowledge on nodes

Why decentralization?

Limitations of a central vision

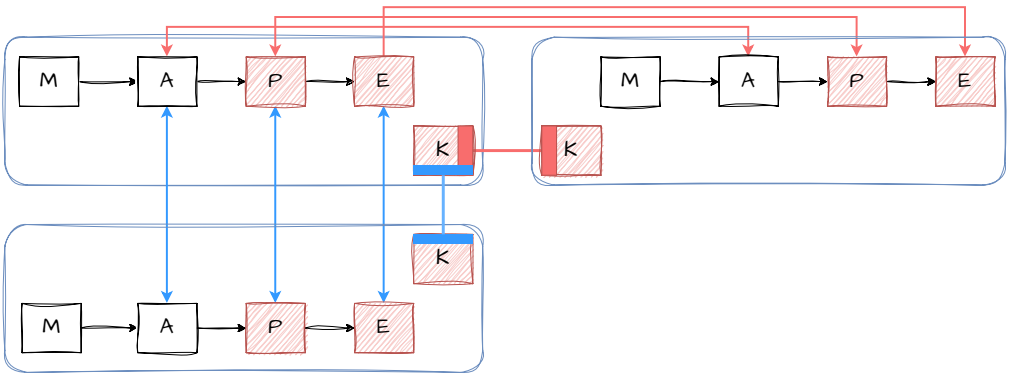


Concerto

- Central entity with the full knowledge
- Nodes simply receive the tasks of a transition
- No knowledge on nodes

- Single point of failure, no local progress
- All blocked in inconsistent state

Ballet

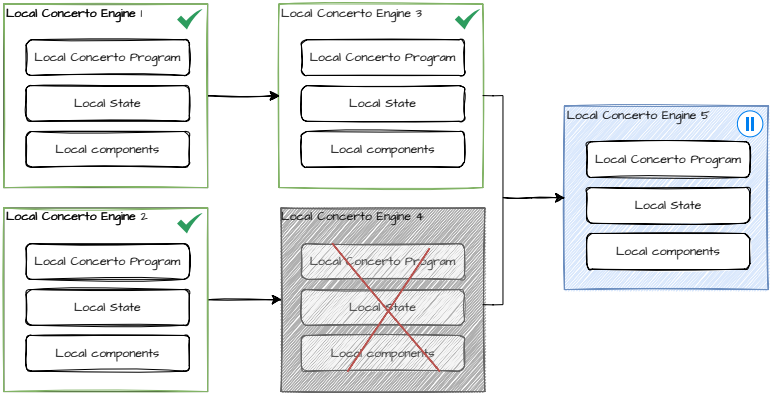


Ballet = Decentralized P & E (declarative)

Full decentralized MAPE-K: ANR SeMaFoR (WP leader, led by Thomas Ledoux)

Concerto-D

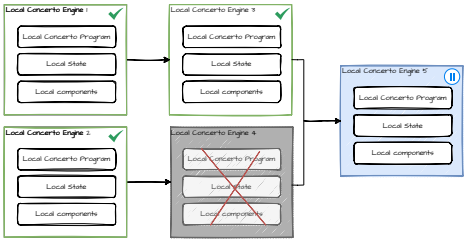
Decentralized Concerto



Local progress is possible + less to achieve when the node is back

CONCERTO-D

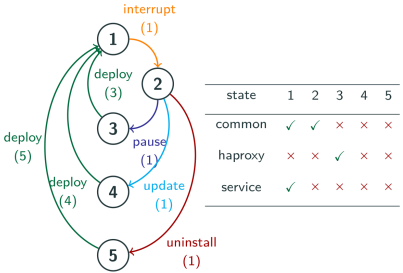
DECENTRALIZED CONCERTO



Concerto-D vs Concerto

- local `add/del`
- local `con/dcon`
  - communications: synchronized `dcon`
- local `pushB`
  - identified `pushB`
  - communications: statuses of ports
- `wait` for local or distant components
  - communications: end of behaviors

DECENTRALIZED PLANNER



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

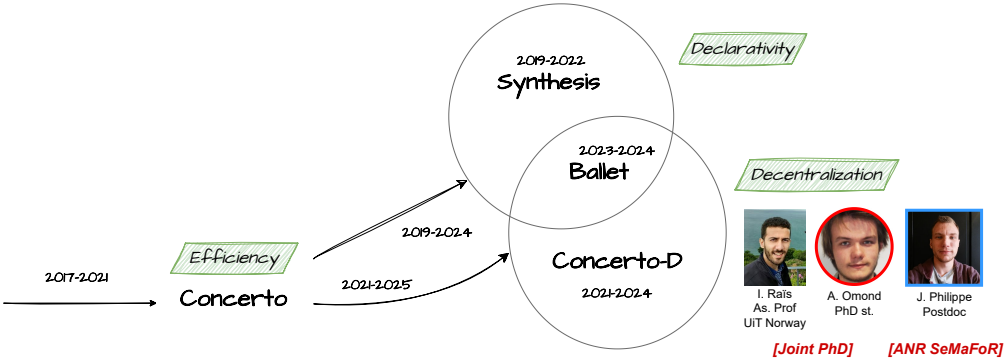
Artifacts on Grid'5000

- Local constraint programming model
- Propagation with gossip-like algorithm
- Received messages enrich local models
  - additional behaviors
  - `wait` instructions
- Final consensus: end of the propagation
- Execution is started with Concerto-D

Deployment and update of multi-site OpenStack $[1, 2, 5, 10]$ sites

- 40% (deployment) and 24% (update) gain compared to Mjuz
- Planning time is less than 2% of the execution time
- 10 sites: ∼200 constraints and ∼100 instructions inferred

Dancers

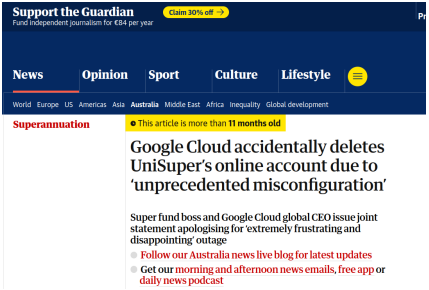


Jolan Philippe, Antoine Omond, Hélène Coullon, Charles Prud'homme, Issam Rais. *Fast Choreography of Cross-DevOps Reconfiguration with Ballet: A Multi-Site OpenStack Case Study*. In IEEE SANER 2024, Finland

Safety

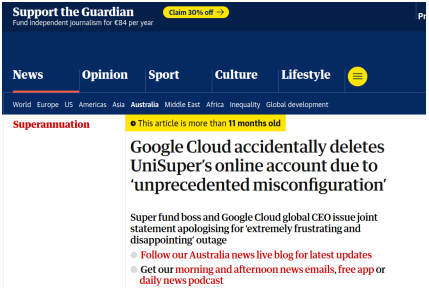
- Why safety?
- Verification
- Formal specification
- Conductors

WHY SAFETY?



"Google Cloud CEO, Thomas Kurian has confirmed that the disruption arose from an unprecedented sequence of events whereby an inadvertent misconfiguration during provisioning of UniSuper's Private Cloud services ultimately resulted in the deletion of UniSuper's Private Cloud subscription," the pair said.

Why safety?


Support the Guardian
Fund independent journalism for €84 per year
Claim 30% off →
News Opinion Sport Culture Lifestyle
World Europe US Americas Asia Australia Middle East Africa Inequality Global development
Superannuation
This article is more than 11 months old
Google Cloud accidentally deletes UniSuper's online account due to 'unprecedented misconfiguration'
Super fund boss and Google Cloud global CEO issue joint statement apologising for 'extremely frustrating and disappointing' outage
Follow our Australia news live blog for latest updates
Get our morning and afternoon news emails, free app or daily news podcast

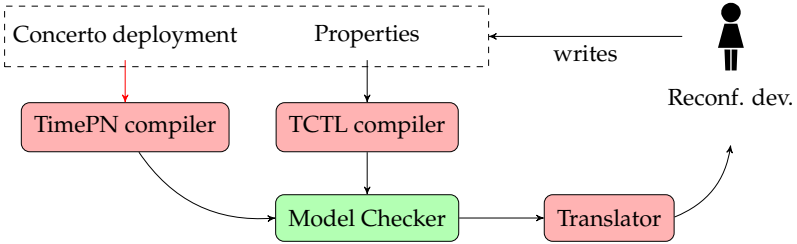
"Google Cloud CEO, Thomas Kurian has confirmed that the disruption arose from an unprecedented sequence of events whereby an inadvertent misconfiguration during provisioning of UniSuper's Private Cloud services ultimately resulted in the deletion of UniSuper's Private Cloud subscription," the pair said.

Services and infrastructures configurations are critical (day-to-day organization, health, energy, network, etc.). Configuration is difficult, particularly in Concerto.

Formal methods: mathematically rigorous techniques for specifying, verifying and synthesizing software systems

Verification of a deployment

Restricted to initial deployments



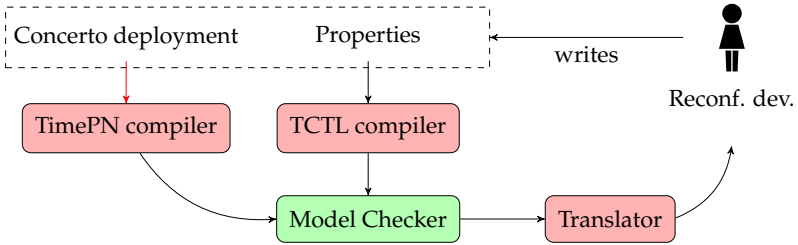
Qualitative properties

- `deployability` (inevitability)
- `sequentiality` (observer subnet)
- `forbidden` (observer subnet)

+ quantitative properties

Verification of a deployment

Restricted to initial deployments



Qualitative properties

- `deployability` (inevitability)
- `sequentiality` (observer subnet)
- `forbidden` (observer subnet)

+ quantitative properties

Scalablity issues

- Unfeasible for (full) Concerto programs
- Same issue with FOL and SMT

→ What about synthesizing programs instead?

Formal specification of Concerto and Concerto-D

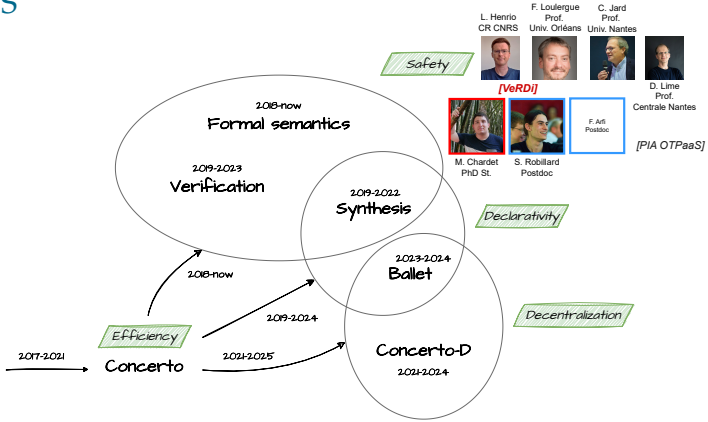
Formal specification of the language (instead of a program)

- Ambiguities (problems) can be discovered and resolved in the language
- Mathematical reference to guide development
- Basis to verify the correctness of any program

Operational semantics of Concerto and Concerto-D

- Pen-and-paper semantics of Concerto
- Mechanized semantics of Concerto/Concerto-D
  - Maude rewriting system
  - Executability of the semantics

Conductors

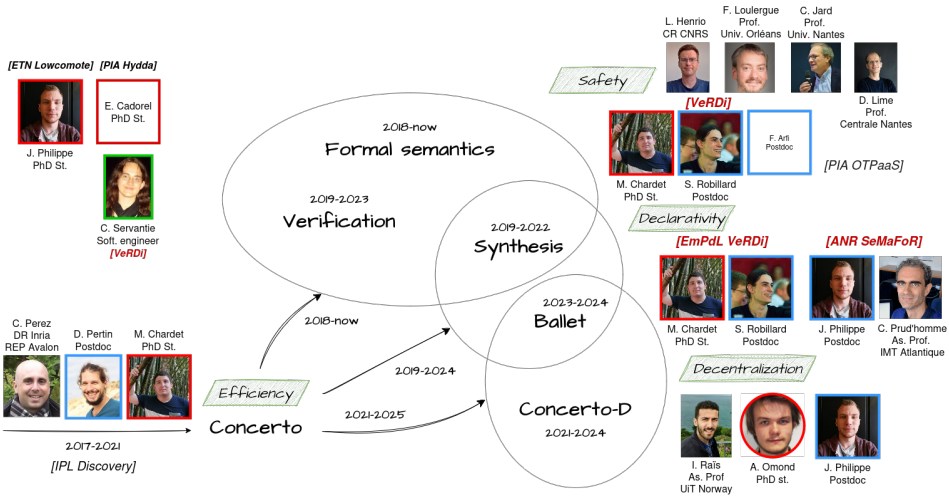


- Farid Arfi, Hélène Coullon, Frédéric Loulergue, Jolan Philippe, Simon Robillard. A Maude Formalization of the Distributed Reconfiguration Language Concerto-D. In ICE@DisCoTeC 2024, Groningen, The Netherlands
- Simon Robillard, Hélène Coullon. *SMT-Based Planning Synthesis for Distributed System Reconfigurations.* In FASE 2022, Munich, Germany
- Maverick Chardet, Hélène Coullon, Simon Robillard. *Toward Safe and Efficient Reconfiguration with Concerto.* SCP, 2021
- Hélène Coullon, Didier Lime, Claude Jard. *Integrated Model-checking for the Design of Safe and Efficient Distributed Software Commissioning.* In iFM 2019, Bergen, Norway

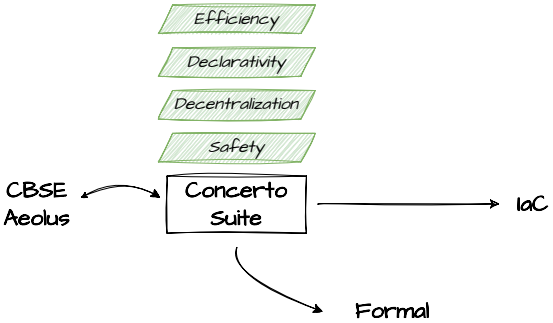
Next

- Conclusion
- From Infrastructure-as-Code to Concerto
- Formally verified Infrastructure-as-Code
- Resilience to crises

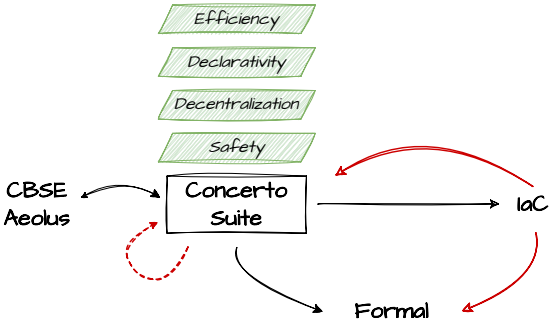
Conclusion


[ETN Lowcomote] [PIA Hydda]
E. Cadorel
PhD St.
J. Philippe
PhD St.
C. Servantie
Soft. engineer
[VeRDi]
2018-now
Formal semantics
2019-2023
Verification
2019-2022
Synthesis
2023-2024
Ballet
2018-now
2019-2024
2021-2025
Efficiency
Concerto
Concerto-D
2021-2024
C. Perez
DR Inria
REP Avalon
D. Pertin
Postdoc
M. Chardet
PhD St.
2017-2021
[IPL Discovery]
Safety
L. Henrio
CR CNRS
F. Loulergue
Prof.
Univ. Orléans
C. Jard
Prof.
Univ. Nantes
D. Lime
Prof.
Centrale Nantes
[VeRDi]
M. Chardet
PhD St.
S. Robillard
Postdoc
F. Arfi
Postdoc
[PIA OTPaaS]
Declarativity
[EmPDL VeRDi]
[ANR SeMaFoR]
M. Chardet
PhD St.
S. Robillard
Postdoc
J. Philippe
Postdoc
C. Prud'homme
As. Prof.
IMT Atlantique
Decentralization
I. Raïs
As. Prof
UiT Norway
A. Omond
PhD st.
J. Philippe
Postdoc
[Joint PhD]
[ANR SeMaFoR]

Conclusion

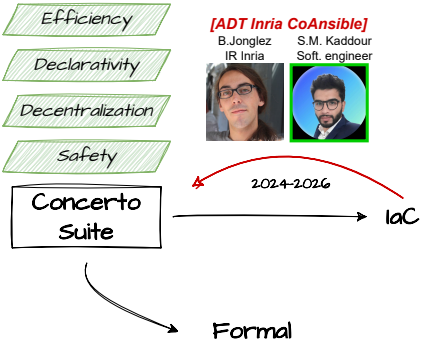

Efficiency
Declarativity
Decentralization
Safety
CBSE
Aeolus
Concerto
Suite
IaC
Formal

Conclusion


Efficiency
Declarativity
Decentralization
Safety
CBSE
Aeolus
Concerto
Suite
IaC
Formal

From Infrastructure-as-Code to Concerto

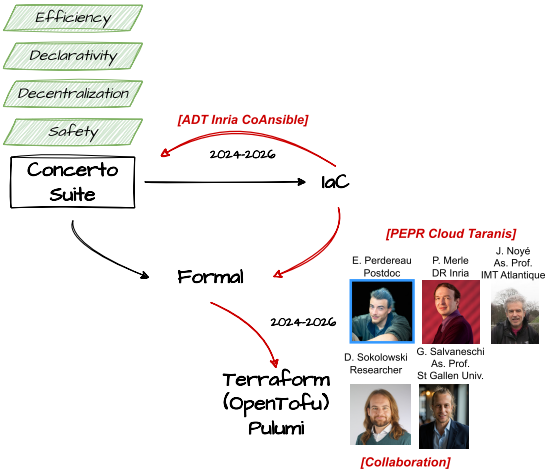
Most IaC languages manipulate life cycles of resources → why not leverage Concerto?



CoAnsible: Transfer Action Inria

- Concerto as a library/API
- Ansible extension (new coordinator)
- Additional YAML file
  - inputs/outputs of roles
  - bindings of inputs/outputs to tasks

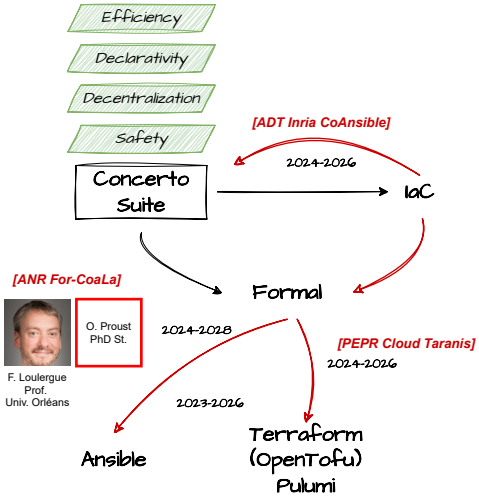
Formally Verified Infrastructure-as-Code



Formal semantics and verification provisioning tools

- formal semantics of Terraform in Maude
- generalization to other provisioning tools
- verify properties (e.g., dep. stability)
- introducing programmable life cycles in provisioning tools

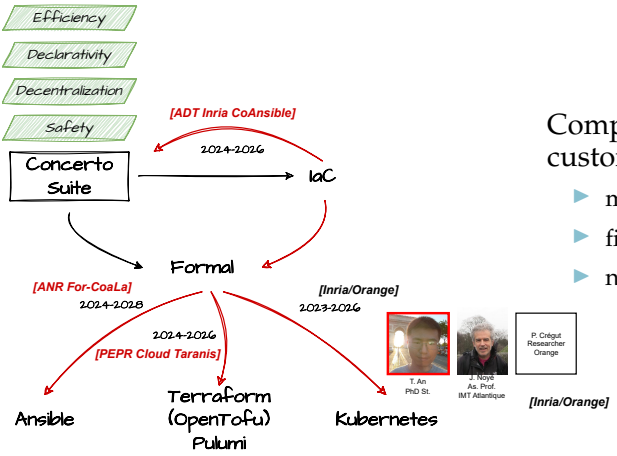
Formally Verified Infrastructure-as-Code



Formal semantics and verification of Ansible

- widely used tool with peculiar semantics
- formal semantics of Ansible in Coq
- verify properties (e.g., idempotency)
- similar work on CoAnsible

Formally Verified Infrastructure-as-Code



Composition issue with Kubernetes custom controllers (operators)

- model the problem
- find properties for good composition
- new methodologies to write operators

Resilience to crisis

New Inria team under construction

- Infrastructure is more and more prominent in our daily life, including critical services (health, banks, industries, telecommunications, energy, etc.)
- Infrastructure is subject to more and more crises

Crises

- Climate change:
  - extreme conditions: humidity, heatwave, flooding, fires, etc.
- Geopolitical instability and sovereignty
  - uncertainties on metals, energy, hardware, network, software etc.
- Depletion of natural resources
  - limits on metals, water, energy, etc.

Resilience to crises

Find new ways to design and operate infrastructures under crisis

Proactive mechanisms

- Local-first
- Delay-tolerant
- Limits-by-design
- Heterogeneity
- Refurbished hardware
- etc.

Reactive mechanisms

- Reconfiguration/IaC
- Approximate comp.
- Urgent computing
- Graceful degradation
- Scheduling policies
- etc.

Thank you for your attention!