Reversible Computation vs. Runtime Adaptation

(in Industrial IoT Systems)

CORSE 2025 · D. P. Attard · K. Bugeja · A. Francalanza · M. Galea · G. Tabone · G. Zahra

University of Malta

Introduction

Smart factories consist of **roboticised** shop floors

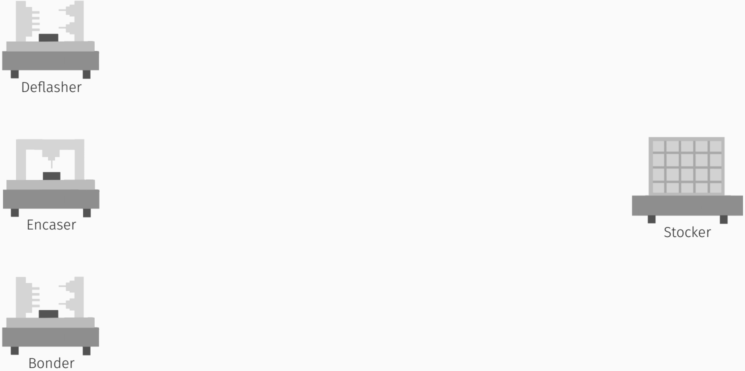
IIoT enables **real-time** monitoring and control

IIoT computation resources per device are **limited**

Automation requirements

- **Robust**: anticipate errors ⇒ **safe** production process
- **Flexible**: adapt to errors ⇒ **uninterrupted** production process

Example: IC manufacturing shop floor



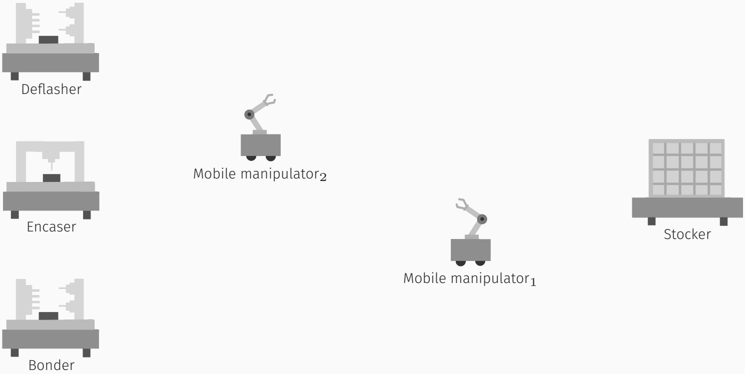
Deflasher

Encaser

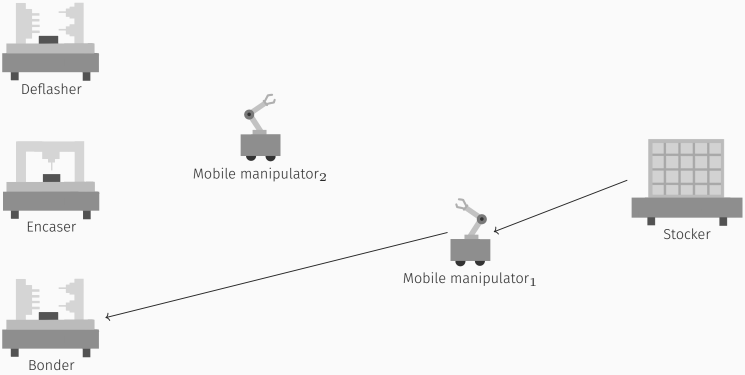
Bonder

Stocker

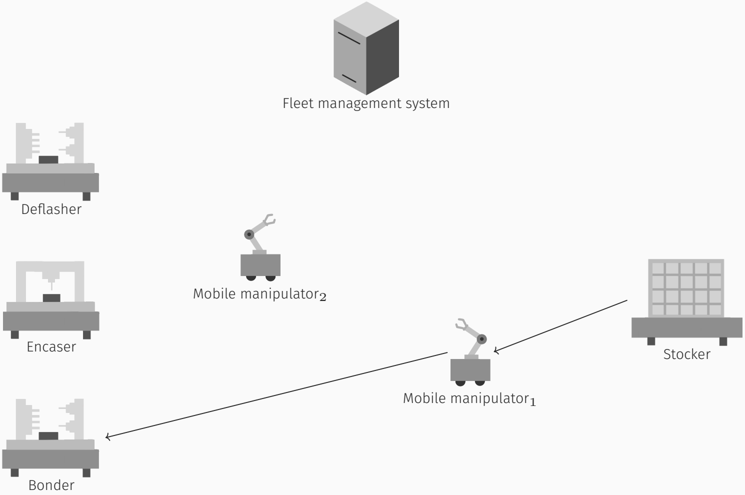
Example: IC manufacturing shop floor



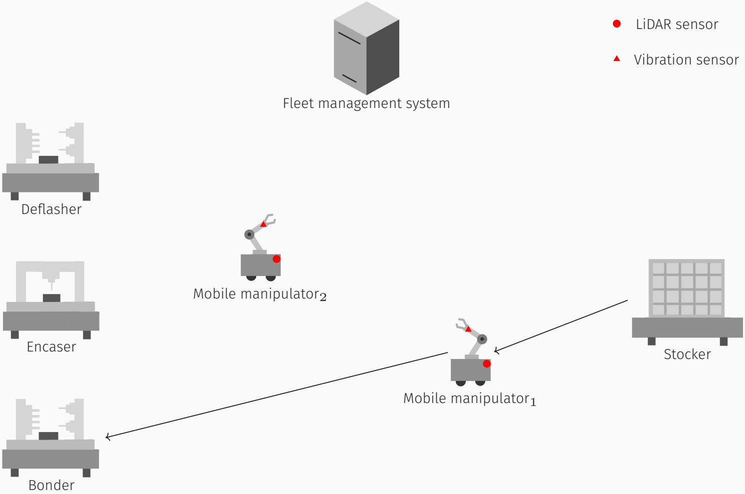
Example: IC manufacturing shop floor



Example: IC manufacturing shop floor



Example: IC manufacturing shop floor



● LiDAR sensor

▲ Vibration sensor

Fleet management system

Deflasher

Encaser

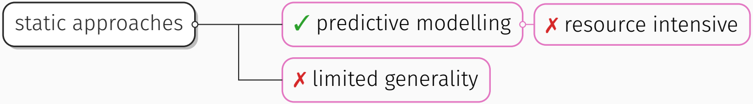
Bonder

Mobile manipulator$_2$

Mobile manipulator$_1$

Stocker

Automation robustness and flexibility



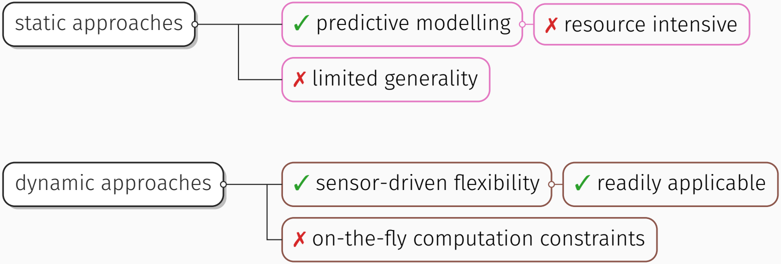
static approaches

✓ predictive modelling

✗ resource intensive

✗ limited generality

Automation robustness and flexibility



static approaches

✓ predictive modelling

✗ resource intensive

✗ limited generality

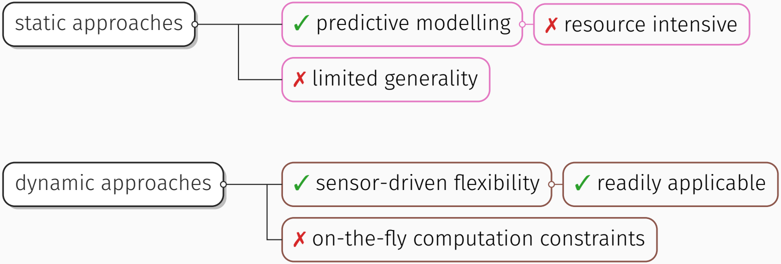
dynamic approaches

✓ sensor-driven flexibility

✓ readily applicable

✗ on-the-fly computation constraints

Automation robustness and flexibility



Balancing static and dynamic approaches

- Use part of the **model** information to reduce hardware costs
- Use **sensor** information to adapt to changes as they occur

Runtime adaptation (RA)

Detects **violations** of correctness properties

Monitors respond with remedial actions (called **adaptations**)

Applicable to **black-box** systems with limited internal access

A RA correctness property

- Is formalised **declaratively**, *e.g.*, via a temporal logic
- Encodes **static knowledge** about the system
- Defines the set of **error states** the system can be in

Reversible computation (RC)

Partitions computations as **forwards** and **backwards**

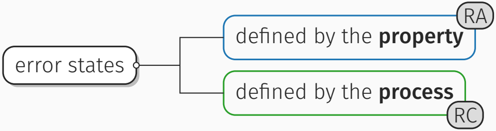
Programs can **undo** computation via backward steps

Requires **intimate knowledge** of and **access** to system code

Reversibility can be

- **Direct:** an inverse action of the forward action exists
- **Indirect:** backward actions needed to reverse a forward action
- **Irreversible:** no direct or indirect backward action exists

RA and RC side-by-side



error states

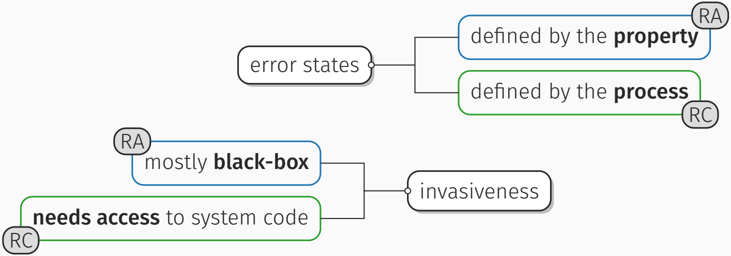
defined by the **property**

RA

defined by the **process**

RC

RA and RC side-by-side



error states

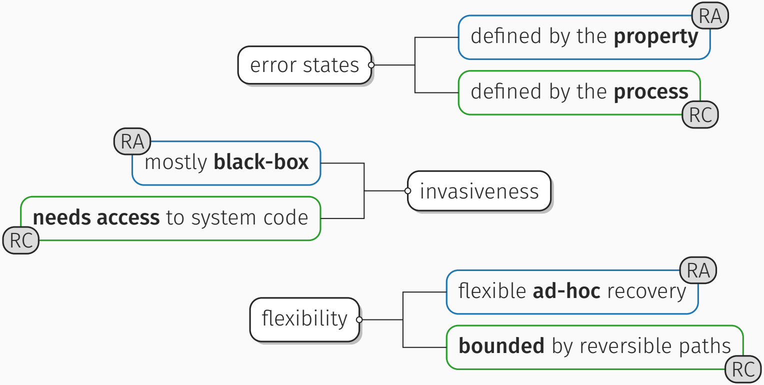
- defined by the **property** RA
- defined by the **process** RC

invaseness

invasiveness

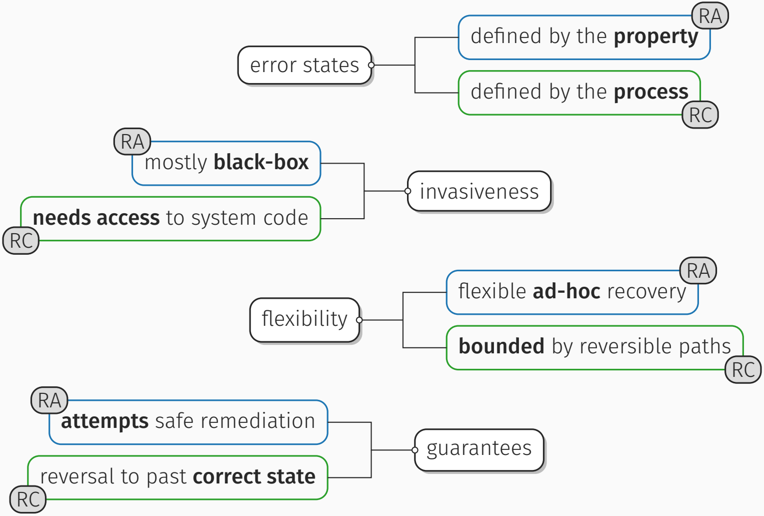
- RA mostly **black-box**
- RC **needs access** to system code

RA and RC side-by-side



- error states
  - defined by the **property** RA
  - defined by the **process** RC
- invasiveness
  - RA mostly **black-box**
  - **needs access** to system code RC
- flexibility
  - flexible **ad-hoc** recovery RA
  - **bounded** by reversible paths RC

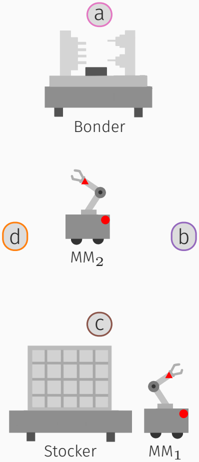
RA and RC side-by-side



- error states
  - RA: defined by the **property**
  - RC: defined by the **process**
- invasiveness
  - RA: mostly **black-box**
  - RC: **needs access** to system code
- flexibility
  - RA: flexible **ad-hoc** recovery
  - RC: **bounded** by reversible paths
- guarantees
  - RA: **attempts** safe remediation
  - RC: reversal to past **correct state**

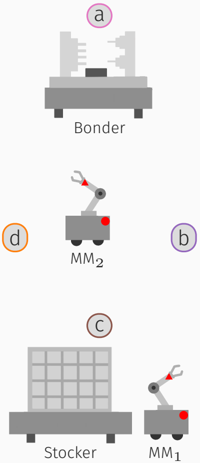
RA and RC in action

> MMs never block when entering a docking station



RA and RC in action

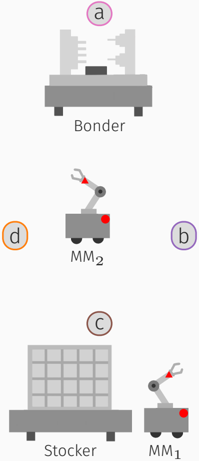
> “ *MMs never block when entering a docking station* ”



Reversible computation

RA and RC in action

> *MMs never block when entering a docking station*

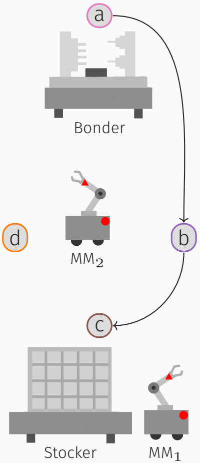


Reversible computation

1. **pick**$(\text{dev} = \text{MM}_2, \text{from} = \text{Bonder}, \text{obj} = \text{Dies}) \rightarrow$

RA and RC in action

> *MMs never block when entering a docking station*

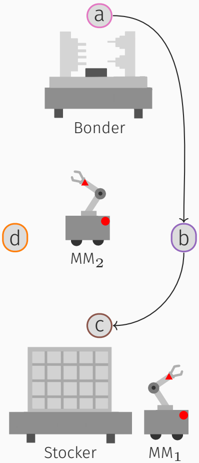


Reversible computation

1. **pick**$(\text{dev} = \text{MM}_2, \text{from} = \text{Bonder}, \text{obj} = \text{Dies}) \rightarrow$
2. **move**$(\text{dev} = \text{MM}_2, \text{from} = \text{Bonder}, \text{to} = \text{Stocker}, \text{waypts} = [a, b, c]) \rightarrow$

RA and RC in action

❝ *MMs never block when entering a docking station* ❞



Reversible computation

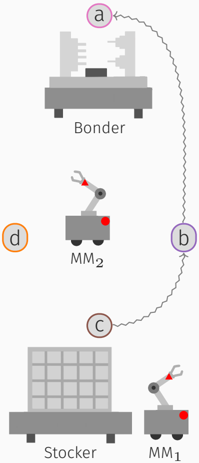
1. **pick**(dev = MM$_2$, from = Bonder, obj = Dies) →

2. **move**(dev = MM$_2$, from = Bonder, to = Stocker, waypts = [a, b, c]) →

3. **blok**(dev = MM$_2$, at = Stocker)

RA and RC in action

> “ *MMs never block when entering a docking station* ”

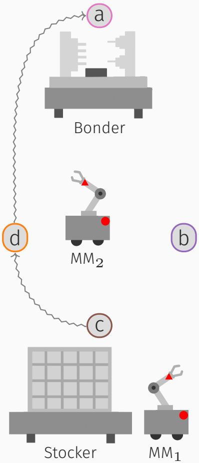


Reversible computation

1. **pick**(dev = MM_2, from = Bonder, obj = Dies) →
2. **move**(dev = MM_2, from = Bonder, to = Stocker, waypts = [a,b,c]) →
3. **move**(dev = MM_2, from = Stocker, to = Bonder, waypts = [c,b,a]) ↜

RA and RC in action

> “ *MMs never block when entering a docking station* ”

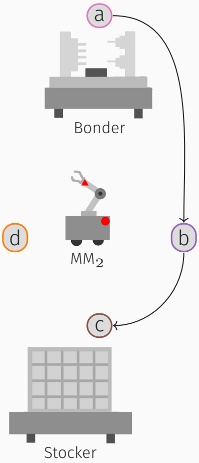


Reversible computation

1. **pick**(dev = MM$_2$, from = Bonder, obj = Dies) →
2. **move**(dev = MM$_2$, from = Bonder, to = Stocker, waypts = [a,b,c]) →
3. **move**(dev = MM$_2$, from = Stocker, to = Bonder, waypts = [c,d,a]) $\leftsquigarrow$

RA and RC in action

> “ *MMs never block when entering a docking station* ”

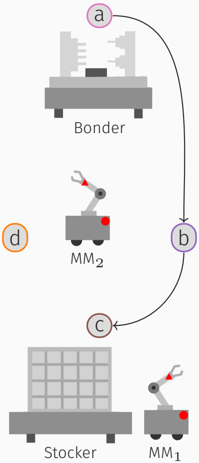


Reversible computation

1. **pick**(dev = MM$_2$, from = Bonder, obj = Dies) →
2. **move**(dev = MM$_2$, from = Bonder, to = Stocker, waypts = [a, b, c]) →
3. **move**(dev = MM$_2$, from = Stocker, to = Bonder, waypts = [c, d, a]) $\leftsquigarrow$
4. **move**(dev = MM$_2$, from = Bonder, to = Stocker, waypts = [a, b, c]) →

RA and RC in action

> “MMs never block when entering a docking station”

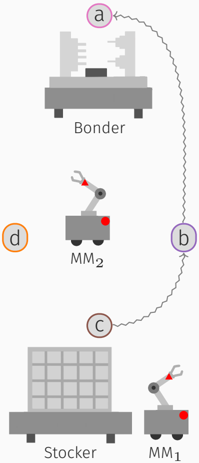


Runtime adaptation

1. **pick**$(\text{dev} = \text{MM}_2, \text{from} = \text{Bonder}, \text{obj} = \text{Dies})$ →
2. **move**$(\text{dev} = \text{MM}_2, \text{from} = \text{Bonder}, \text{to} = \text{Stocker}, \text{waypts} = [a, b, c])$ →
3. **blok**$(\text{dev} = \text{MM}_2, \text{at} = \text{Stocker})$

RA and RC in action

> *❝ MMs never block when entering a docking station ❞*

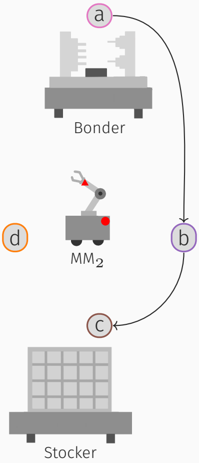


Runtime adaptation

1. **pick**(dev = MM_2, from = Bonder, obj = Dies) $\rightarrow$
2. **move**(dev = MM_2, from = Bonder, to = Stocker, waypts = [a, b, c]) $\rightarrow$
3. **move**(dev = MM_2, from = Stocker, to = Bonder, waypts = [c, b, a]) $\leftsquigarrow$

RA and RC in action

> “MMs never block when entering a docking station”

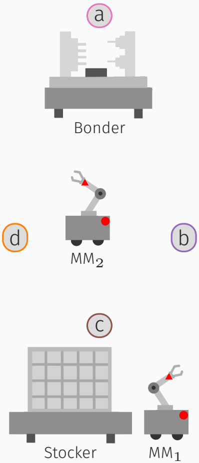


Runtime adaptation

1. **pick**(dev = MM_2, from = Bonder, obj = Dies) →
2. **move**(dev = MM_2, from = Bonder, to = Stocker, waypts = [a,b,c]) →
3. **move**(dev = MM_2, from = Stocker, to = Bonder, waypts = [c,b,a]) $\leftsquigarrow$
4. **move**(dev = MM_2, from = Bonder, to = Stocker, waypts = [a,b,c]) →

RA and RC in action

> *“MMs never block when entering a docking station”*

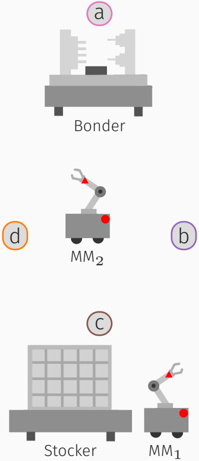


Runtime adaptation

1. **pick**(dev = MM_2, from = Bonder, obj = Dies) →
2. **move**(dev = MM_2, from = Bonder, to = Stocker, waypts = [a, b, c]) →
3. **move**(dev = MM_2, from = Stocker, to = **Stocker′**, waypts = [c, e]) →

RA and RC in action

> *“MMs never block when entering a docking station”*



Runtime adaptation

1. **pick**(dev = MM_2,from = Bonder,obj = Dies) →
2. **move**(dev = MM_2,from = Bonder,to = Stocker,waypts = [a,b,c]) →
3. **move**(dev = MM_2,from = Stocker,to = **Stocker**′,waypts = [c,e]) →

What if the robot arm of MM_2 gets damaged?

Our experience in applying reversibility to IIoT

Machines are black boxes with **limited API**

Machine behaviour must be observed **indirectly** via sensors

Changes in machine behaviour published as **events**

Notion of a bad system state corresponds to an **error event**

Less obvious to identify **forward** and **backward** computations

RA seems to be more applicable to our IIoT setting

How can we benefit from RC?

Reversible programs via RC

RC gives **static guarantees** about reversible operations

but...

Requires **explicit reasoning** on forward and backward logic

Tightly couples forward and backward logic in code

May be harder to reverse **side-effecting** operations

The RA view of reversibility

Treat reversibility as **cross-cutting**

Separation of concerns between forward and backward logic

Automate the generation of a complete reversible program

Benefits

- **Simplicity:** reason on forward and backward logic separately
- **Modularity:** backward logic can be layered as RA actions
- **Static guarantees:** when RA is limited to reversible operations

Reversible programs via RA

RA property φ

1. RA adaptation defines backward logic

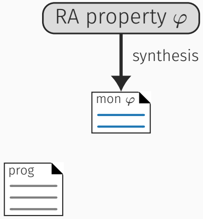
Reversible programs via RA

RA property φ

1. RA adaptation defines backward logic
2. Program defines forward logic

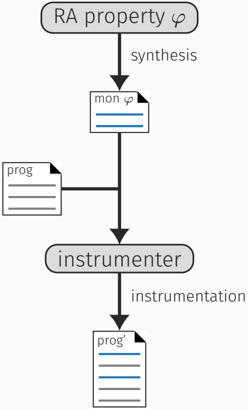
prog

Reversible programs via RA



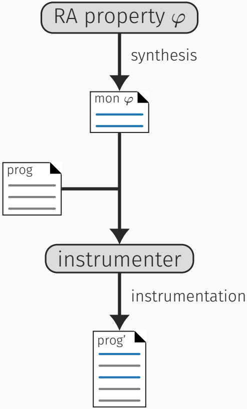
1. RA adaptation defines backward logic
2. Program defines forward logic
3. RA property synthesised as monitor

Reversible programs via RA



1. RA adaptation defines backward logic
2. Program defines forward logic
3. RA property synthesised as monitor
4. Program instrumented with monitor

Reversible programs via RA



1. RA adaptation defines backward logic
2. Program defines forward logic
3. RA property synthesised as monitor
4. Program instrumented with monitor
5. Outputs reversible program

RA + RC for IIoT: best of both worlds

1. Construct RA properties with **only** reversible operations
2. **Mix** ad-hoc logic with RC operations in RA properties

Integration style 1

- Clean delineation of forward and backward logic
- RA can benefit from RC static guarantees

Integration style 2

- RA can handle cases RC deems irreversible
- RA enables graceful degradation ⇒ automation flexibility

Link to paper