UNIVERSITY OF OSLO

A Self-Adaptive Digital Twin Architecture for Dynamic Resource Management

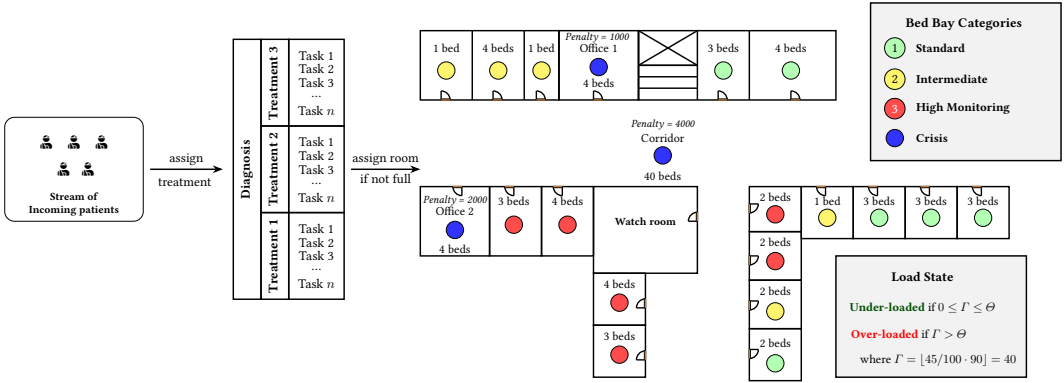
Riccardo Sieve
riccasi@ifi.uio.no
APM Turin 2026

April 16, 2026

Contents

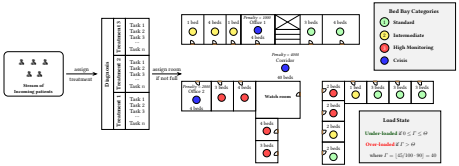
1. Rationale
2. Digital Twin Concepts
3. Digital Twin Architecture
4. Extension to the DT

Why do we need it?

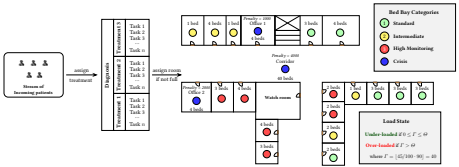


Why do we need it?

- Hospital demand changes quickly; resource capacity does not

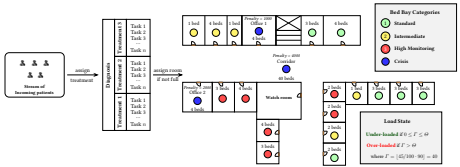


Why do we need it?



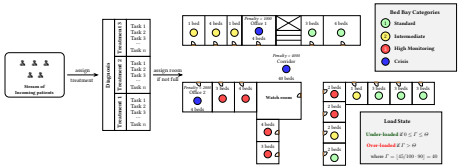
- Hospital demand changes quickly; resource capacity does not
- Admission is often handled manually

Why do we need it?



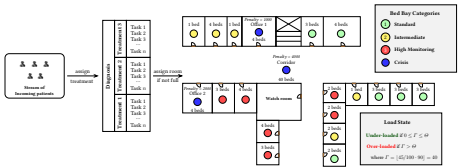
- Hospital demand changes quickly; resource capacity does not
- Admission is often handled manually
- Depends on several factors

Why do we need it?



- Hospital demand changes quickly; resource capacity does not
- Admission is often handled manually
- Depends on several factors
- Static architectures struggle when the underlying structure changes

Why do we need it?



- Hospital demand changes quickly; resource capacity does not
- Admission is often handled manually
- Depends on several factors
- Static architectures struggle when the underlying structure changes
- We need runtime adaptation with trustworthy model alignment

Baseline: BedreFlyt and the gap

- **BedreFlyt** is a hospital-ward digital twin that combines an evolving semantic model, simulation-driven forecasting, and optimisation to recommend bed-bay allocations to staff

Baseline: BedreFlyt and the gap

- **BedreFlyt** is a hospital-ward digital twin that combines an evolving semantic model, simulation-driven forecasting, and optimisation to recommend bed-bay allocations to staff
- It produces bed-bay recommendations for human operators

Baseline: BedreFlyt and the gap

- **BedreFlyt** is a hospital-ward digital twin that combines an evolving semantic model, simulation-driven forecasting, and optimisation to recommend bed-bay allocations to staff
- It produces bed-bay recommendations for human operators
- Human-in-the-loop actions can diverge from DT suggestions

Baseline: BedreFlyt and the gap

- **BedreFlyt** is a hospital-ward digital twin that combines an evolving semantic model, simulation-driven forecasting, and optimisation to recommend bed-bay allocations to staff
- It produces bed-bay recommendations for human operators
- Human-in-the-loop actions can diverge from DT suggestions
- **Main gap**: the original ward layout is treated as fixed

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines
  - Evolving semantic model to represent changing resources at runtime

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines
  - Evolving semantic model to represent changing resources at runtime
  - Lifecycle manager to monitor context and trigger adaptation

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines
  - Evolving semantic model to represent changing resources at runtime
  - Lifecycle manager to monitor context and trigger adaptation
  - Penalty-based optimisation for cost-aware reconfiguration

What is the goal

Goal: enable the digital twin to reconfigure ward capacity at runtime (open/close spare rooms) in response to changing patient demand, while staying aligned with the real ward and keeping a human in the loop.

- A self-adaptive architecture for resource management (BedreFlyt use case) that combines
  - Evolving semantic model to represent changing resources at runtime
  - Lifecycle manager to monitor context and trigger adaptation
  - Penalty-based optimisation for cost-aware reconfiguration
- Human-in-the-loop supervision over the adaptation loop

Contents

1. Rationale
2. Digital Twin Concepts
   - Modelling aspects
3. Digital Twin Architecture
4. Extension to the DT

What is a Digital Twin

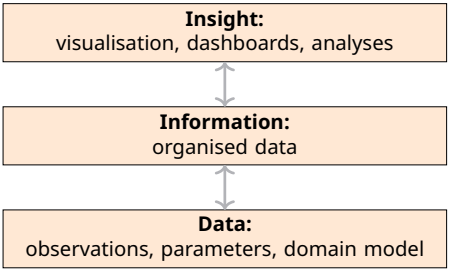
Digital twins are, as defined by NASEM

NASEM Definition of DT (2024)

A digital twin is a set of virtual information constructs that mimics the structure, context, and behaviour of a natural, engineered, or **social system** (or system-of-systems), is dynamically updated with data from its physical twin, has a predictive capability, and **informs decisions** that realise value. The bidirectional interaction between the virtual and the physical is central to the digital twin[a].

[a] National Academies of Sciences, Engineering, and Medicine (NASEM) (2024): Foundational Research Gaps and Future Directions for Digital Twins. The National Academies Press

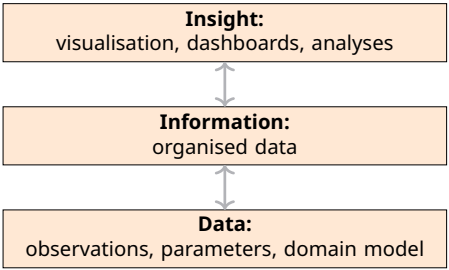
Digital Twins components



Capabilities for different insights:

- **Descriptive:** Insight into the past ("what happened")

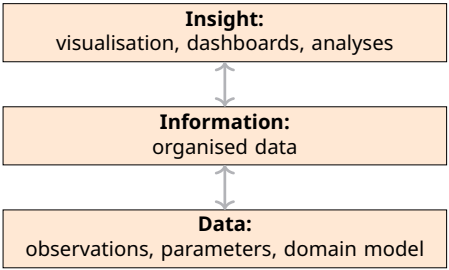
Digital Twins components



Capabilities for different insights:

- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)

Digital Twins components



Capabilities for different insights:

- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)
- **Prescriptive:** Advise on possible outcomes (“what if / what should”)

Digital Twins components

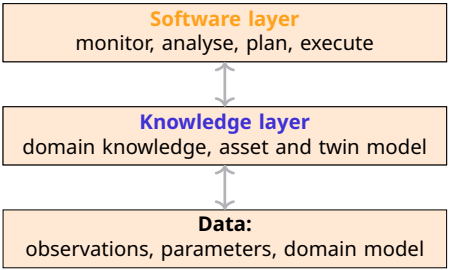
Capabilities for different insights:

- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)
- **Prescriptive:** Advise on possible outcomes (“what if”)

What next:

- **Reactive:** Automated decision making

Digital Twins components



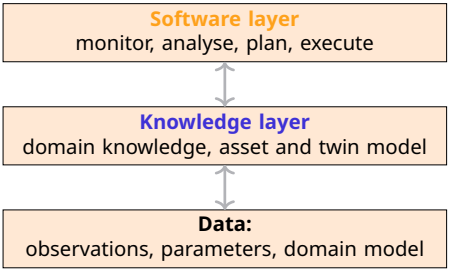
Capabilities for different insights:

- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)
- **Prescriptive:** Advise on possible outcomes (“what if”)

What next:

- **Reactive:** Automated decision making

Digital Twins components



Capabilities for different insights:

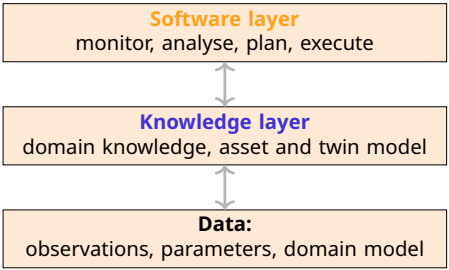
- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)
- **Prescriptive:** Advise on possible outcomes (“what if”)

What next:

- **Reactive:** Automated decision making

- DT must adapt both structurally and behaviourally

Digital Twins components



Capabilities for different insights:

- **Descriptive:** Insight into the past (“what happened”)
- **Predictive:** Understand the future (“what may happen”)
- **Prescriptive:** Advise on possible outcomes (“what if”)

What next:

- **Reactive:** Automated decision making

- DT must adapt both structurally and behaviourally
- States of the DT cycles over time and they need to be captured

Twinning in SMOL

Programming support for twins with a behavioural and a structural layer

smolang.org

Twinning in SMOL

Programming support for twins with a behavioural and a structural layer

SMOL: Semantic Model Object Language

- Smalll OO programming system
- Ontology reasoners allow querying the KB
- Used to create the digital model

smolang.org

Twinning in SMOL

Programming support for twins with a behavioural and a structural layer

smolang.org

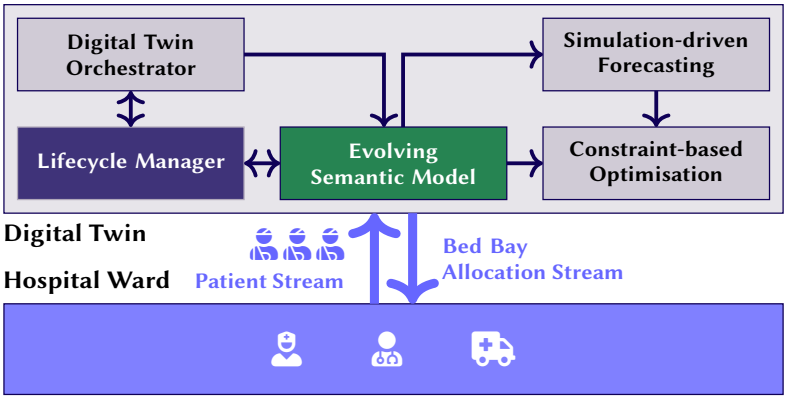
behavioural twins in SMOL

- SMOL can encapsulate (simulation) models based on the FMI standard
- Can automatically adapt the object states in the KB at runtime through semantic lifting

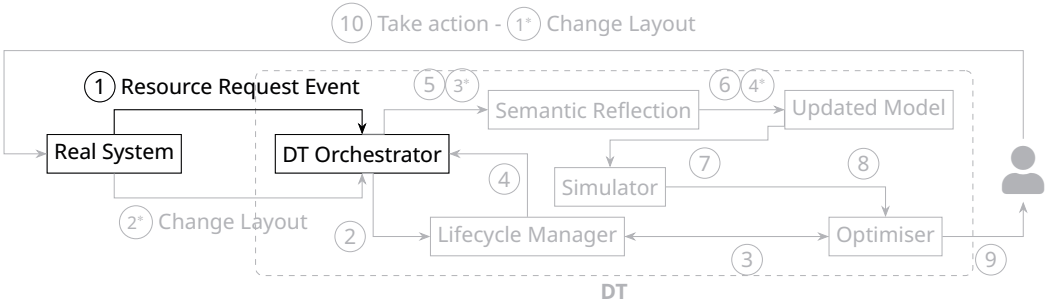
Contents

1. Rationale
2. Digital Twin Concepts
3. Digital Twin Architecture
   - DynResDT
4. Extension to the DT

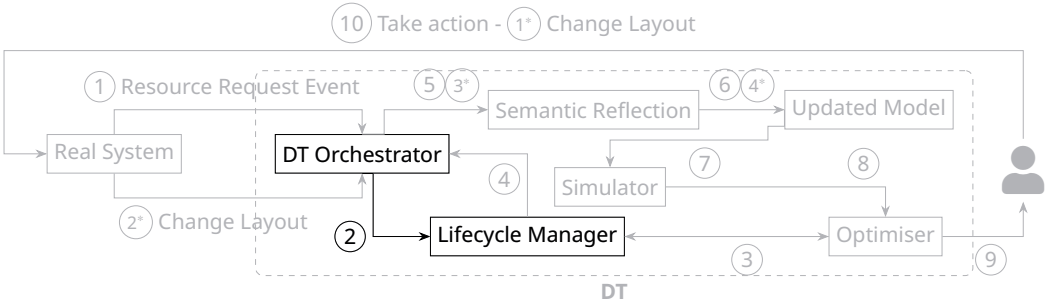
DynResDT: A Self-Adaptive DT Architecture



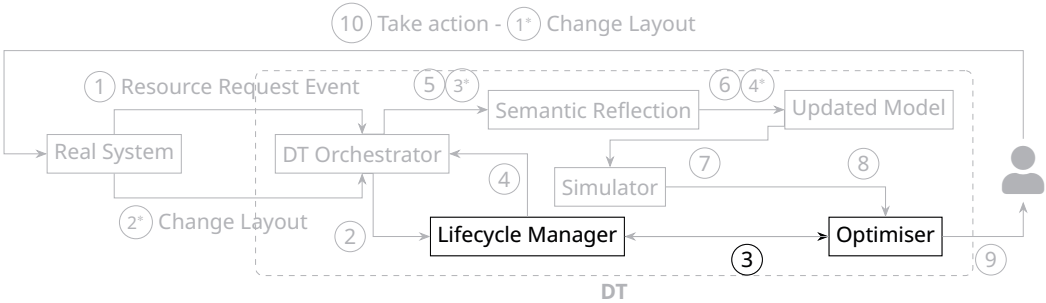
Human-in-the-loop



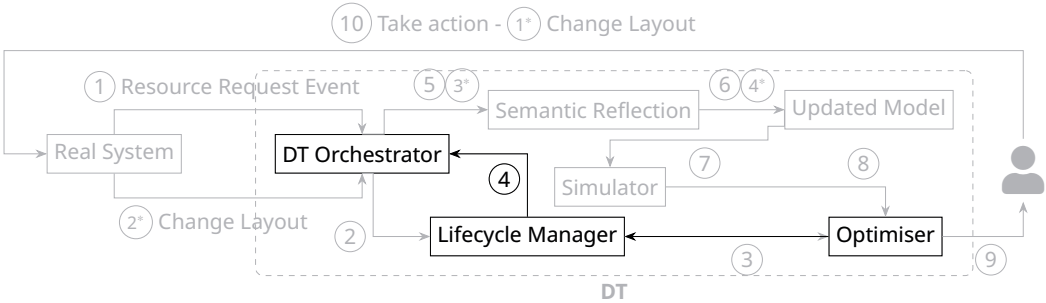
Human-in-the-loop



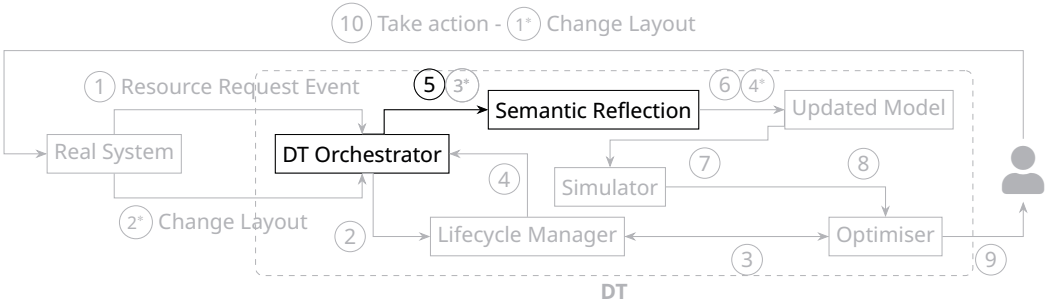
Human-in-the-loop



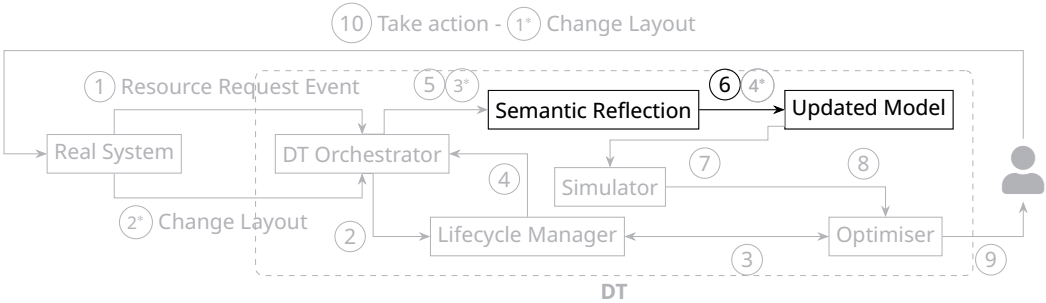
Human-in-the-loop



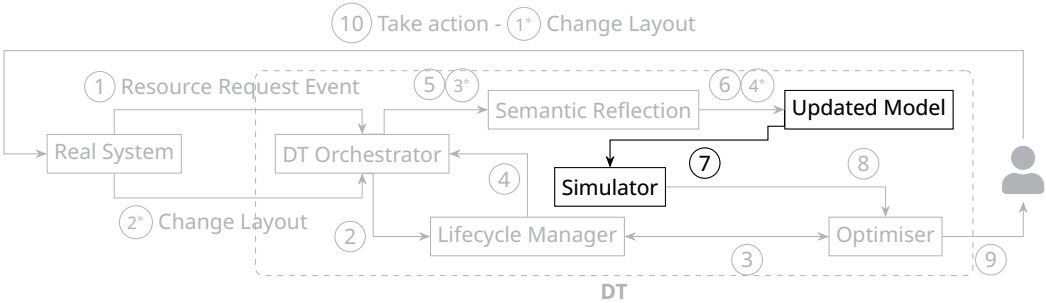
Human-in-the-loop



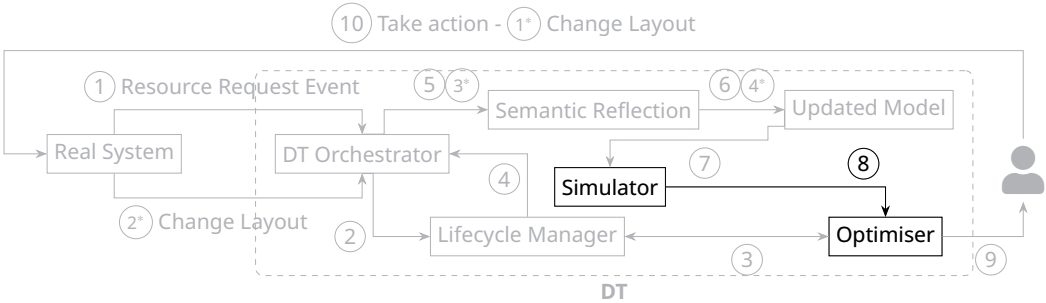
Human-in-the-loop



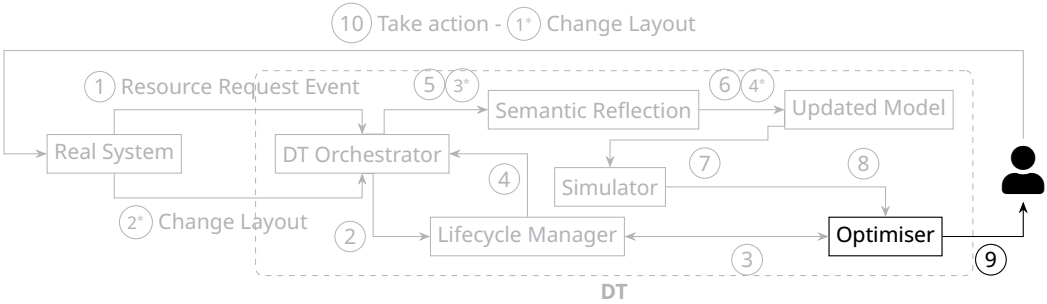
Human-in-the-loop



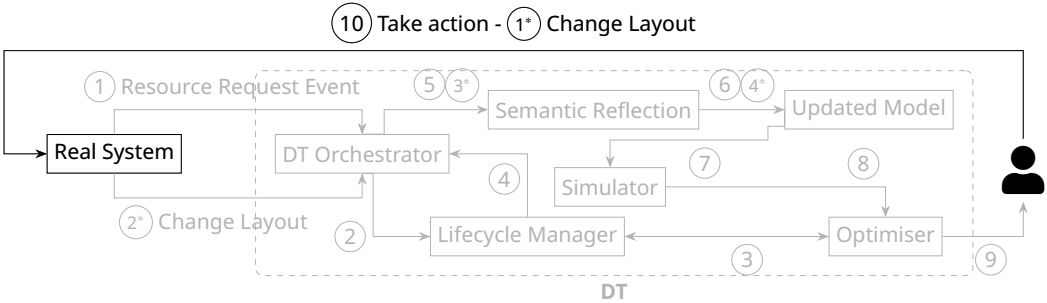
Human-in-the-loop



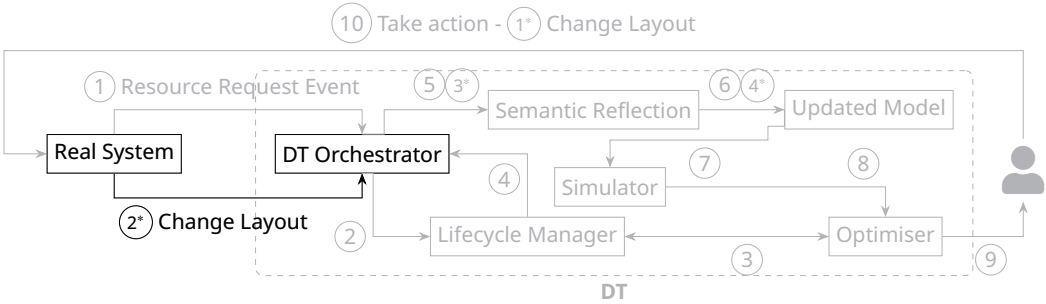
Human-in-the-loop



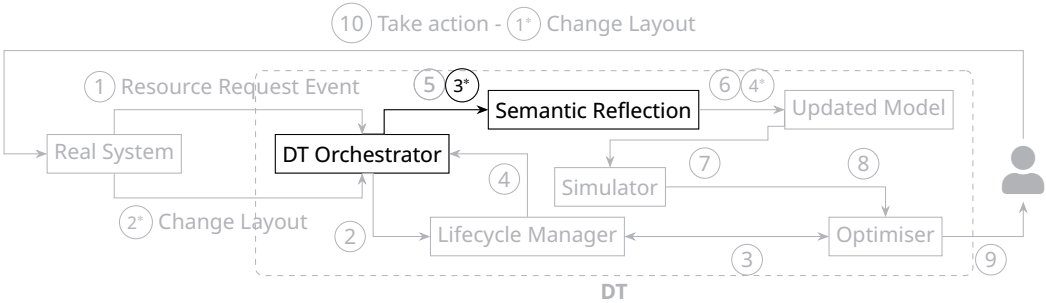
Human-in-the-loop



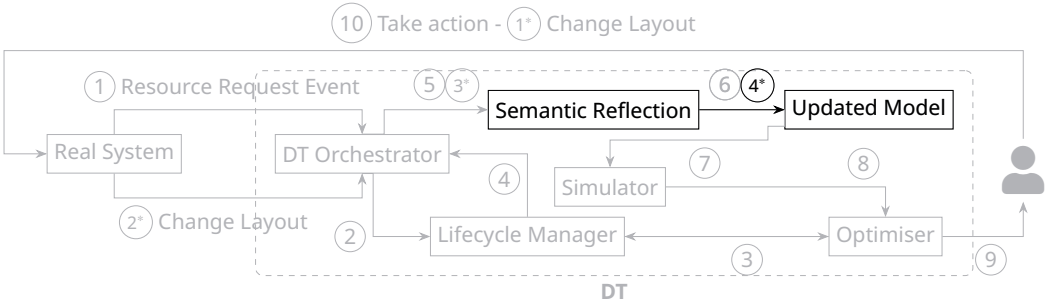
Human-in-the-loop



Human-in-the-loop



Human-in-the-loop



Human-in-the-loop (cont.)

- The DT proposes either *allocation updates* or *layout reconfiguration*

Human-in-the-loop (cont.)

- The DT proposes either *allocation updates* or *layout reconfiguration*
- The human-in-the-loop remains the final authority for adaptation actions

Human-in-the-loop (cont.)

- The DT proposes either *allocation updates* or *layout reconfiguration*
- The human-in-the-loop remains the final authority for adaptation actions
- Approved actions are reflected back via semantic lifting to maintain alignment

Human-in-the-loop (cont.)

- The DT proposes either *allocation updates* or *layout reconfiguration*
- The human-in-the-loop remains the final authority for adaptation actions
- Approved actions are reflected back via semantic lifting to maintain alignment
- Semantic reflection re-aligns the runtime model after structural changes

Human-in-the-loop (cont.)

- The DT proposes either *allocation updates* or *layout reconfiguration*
- The human-in-the-loop remains the final authority for adaptation actions
- Approved actions are reflected back via semantic lifting to maintain alignment
- Semantic reflection re-aligns the runtime model after structural changes
- This enables supervised autonomy: controlled self-adaptation instead of static decision support

Simulation Environment

- To test the allocation process with adaptation we developed an Event Stream Generator (ESG) to create stream of Patients

Simulation Environment

- To test the allocation process with adaptation we developed an Event Stream Generator (ESG) to create stream of Patients
- ESG consists of an *Event Timestamp Generator* (ETG) and an *Event Sample Space* (ESS)

Simulation Environment

- To test the allocation process with adaptation we developed an Event Stream Generator (ESG) to create stream of Patients
- ESG consists of an *Event Timestamp Generator* (ETG) and an *Event Sample Space* (ESS)
- The ETG component generates the timestamp of the event, and the ESS component generates the event instance from a given set of possible event instances

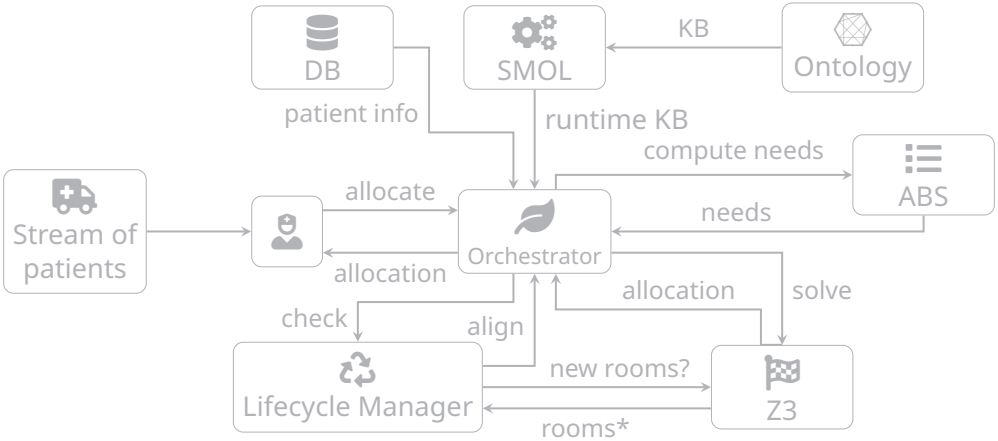
Simulation Environment

- To test the allocation process with adaptation we developed an Event Stream Generator (ESG) to create stream of Patients
- ESG consists of an *Event Timestamp Generator* (ETG) and an *Event Sample Space* (ESS)
- The ETG component generates the timestamp of the event, and the ESS component generates the event instance from a given set of possible event instances
- For every sample, a constraint check is made, and if it does not satisfy the scenario-defined constraint, a resample is triggered

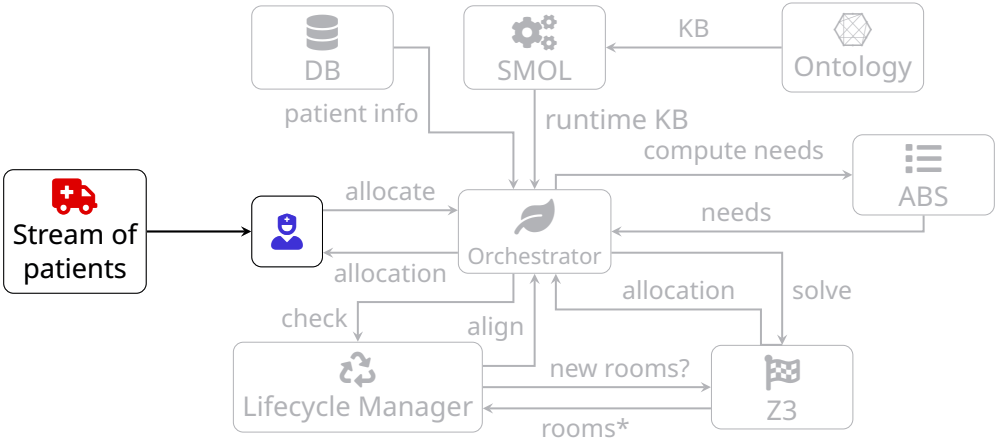
Simulation Environment

- To test the allocation process with adaptation we developed an Event Stream Generator (ESG) to create stream of Patients
- ESG consists of an *Event Timestamp Generator* (ETG) and an *Event Sample Space* (ESS)
- The ETG component generates the timestamp of the event, and the ESS component generates the event instance from a given set of possible event instances
- For every sample, a constraint check is made, and if it does not satisfy the scenario-defined constraint, a resample is triggered
- We allow to account for patient peaks over time, to mimic a period of time of high stress in the ward, switching between high load and low load

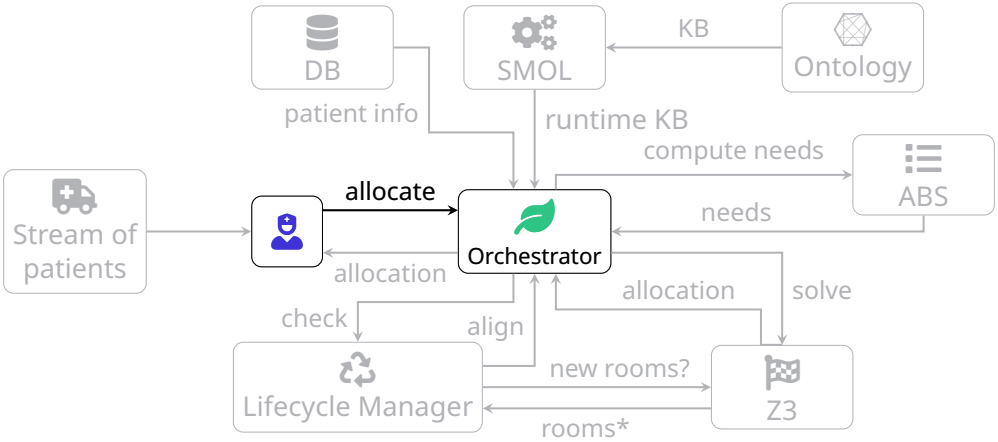
Allocation of patients



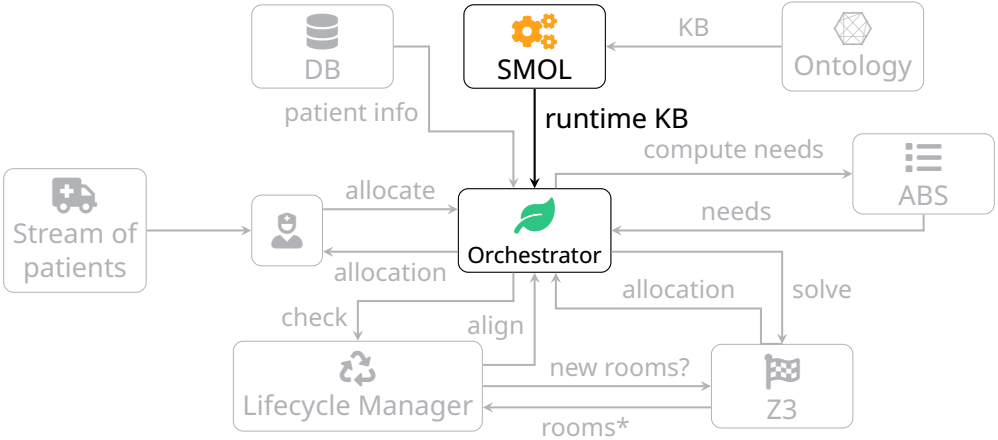
Allocation of patients



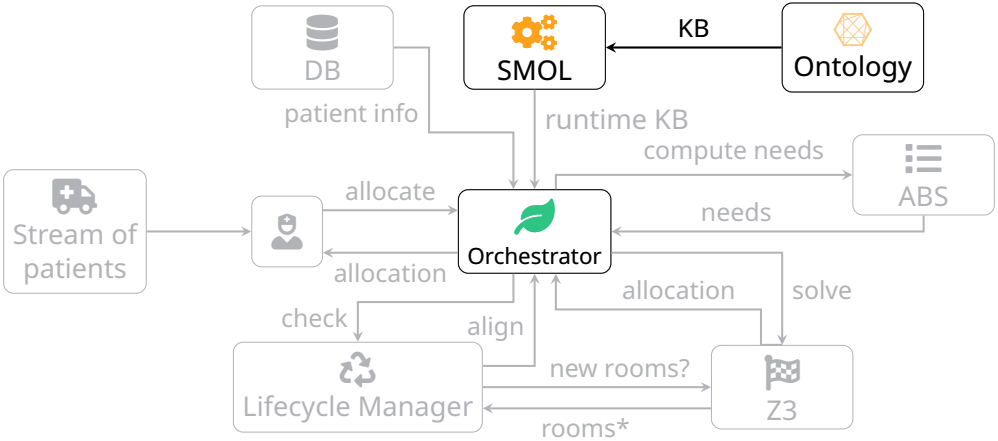
Allocation of patients



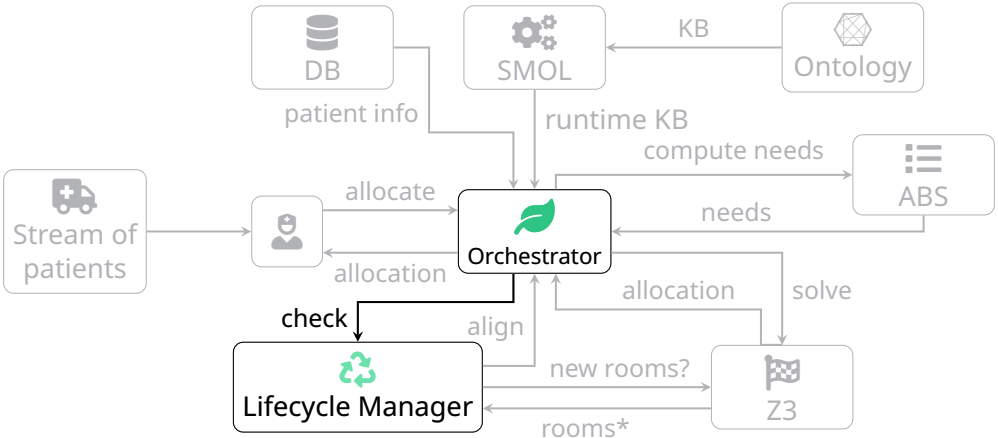
Allocation of patients



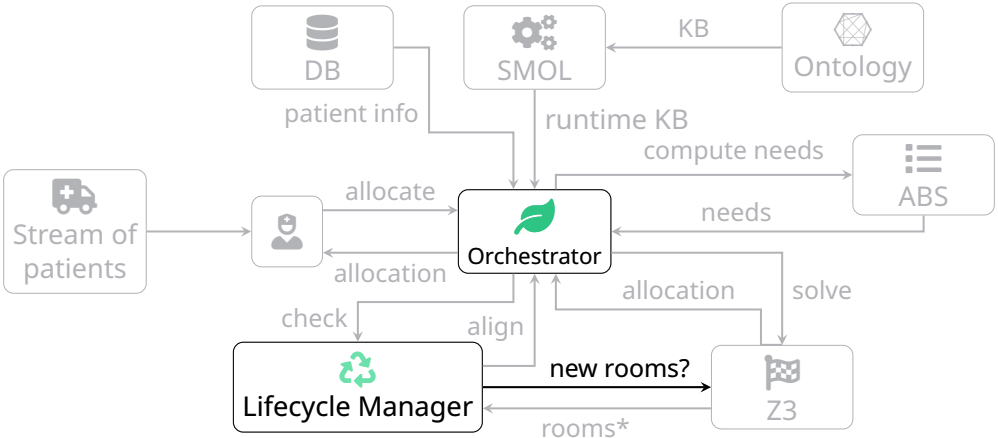
Allocation of patients



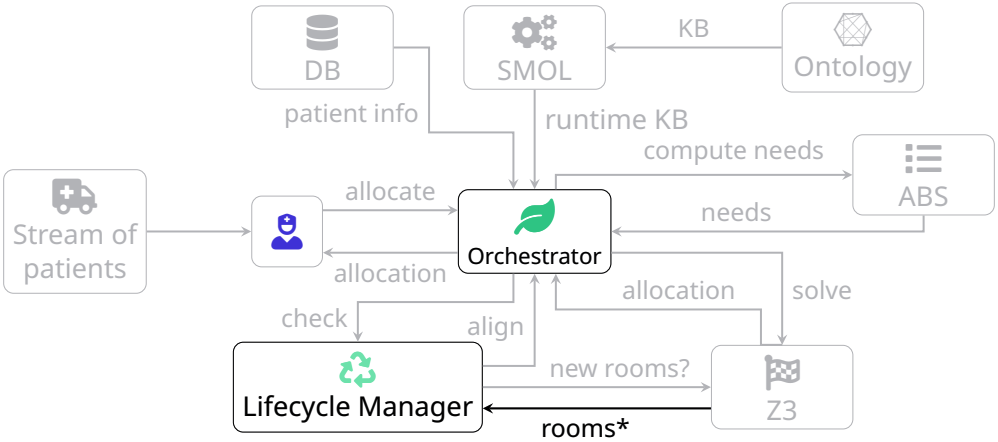
Allocation of patients



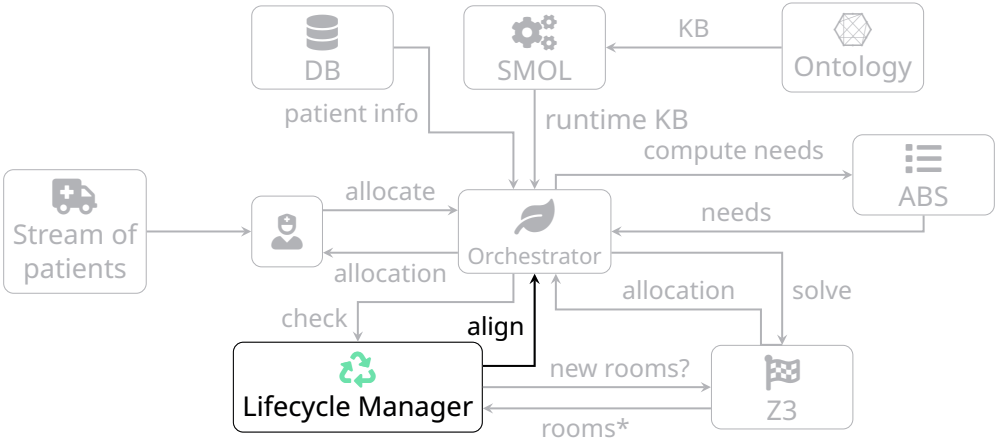
Allocation of patients



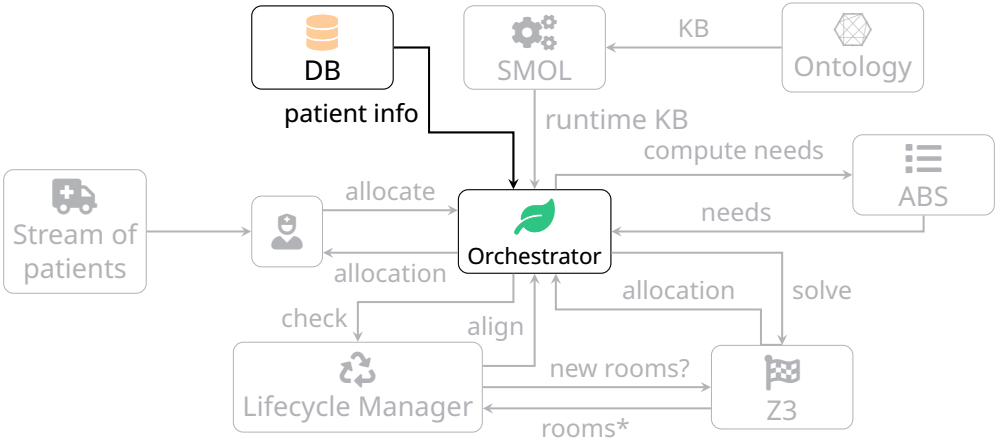
Allocation of patients



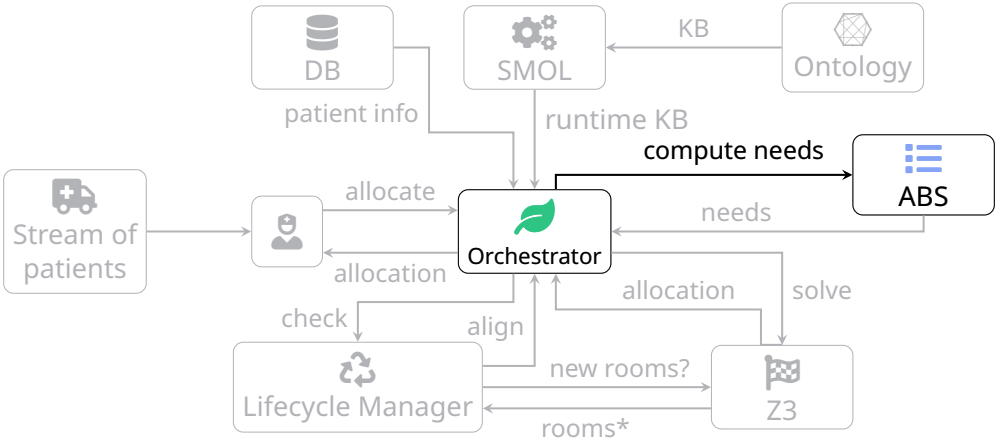
Allocation of patients



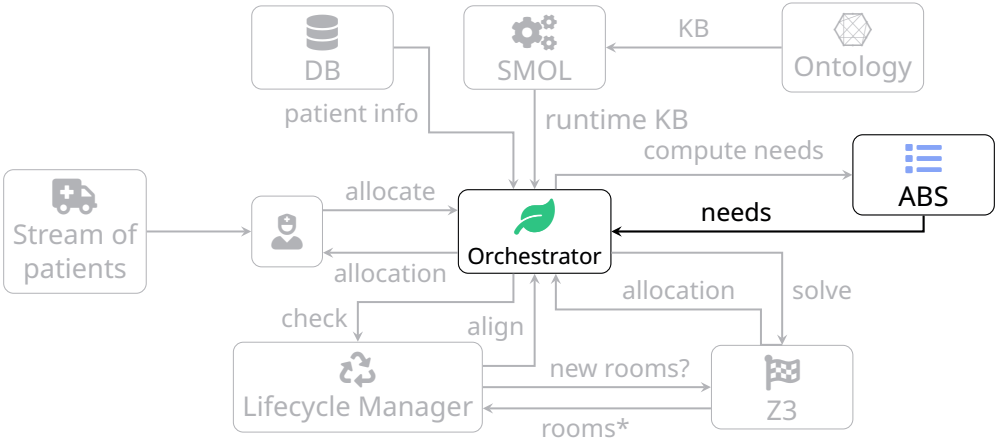
Allocation of patients



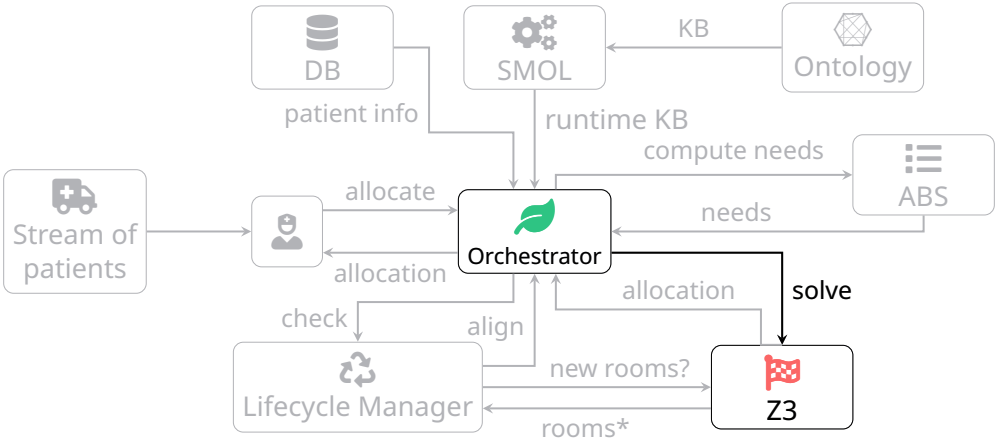
Allocation of patients



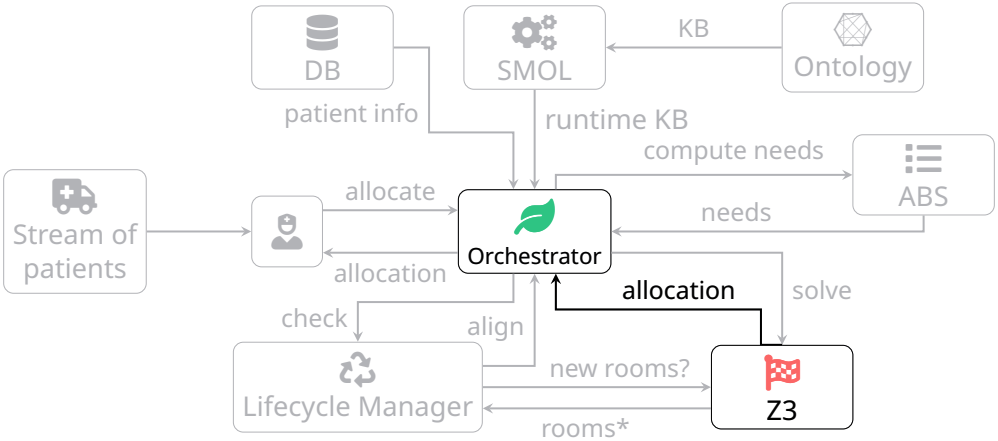
Allocation of patients



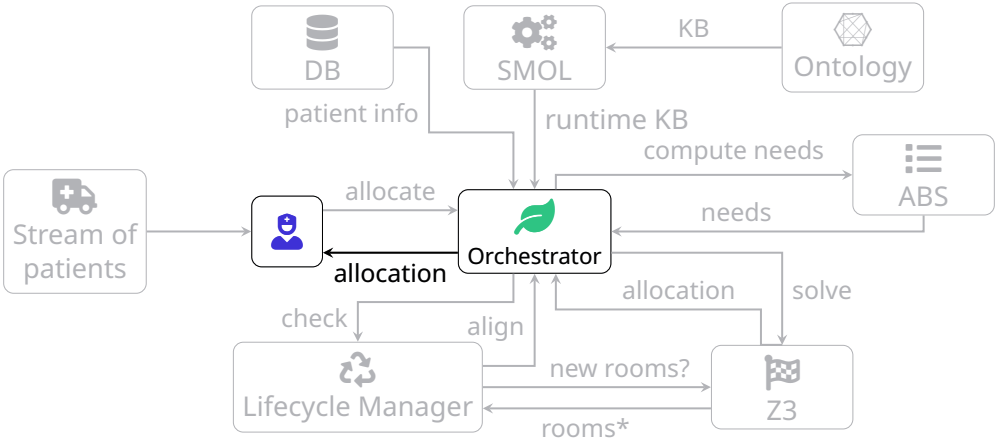
Allocation of patients



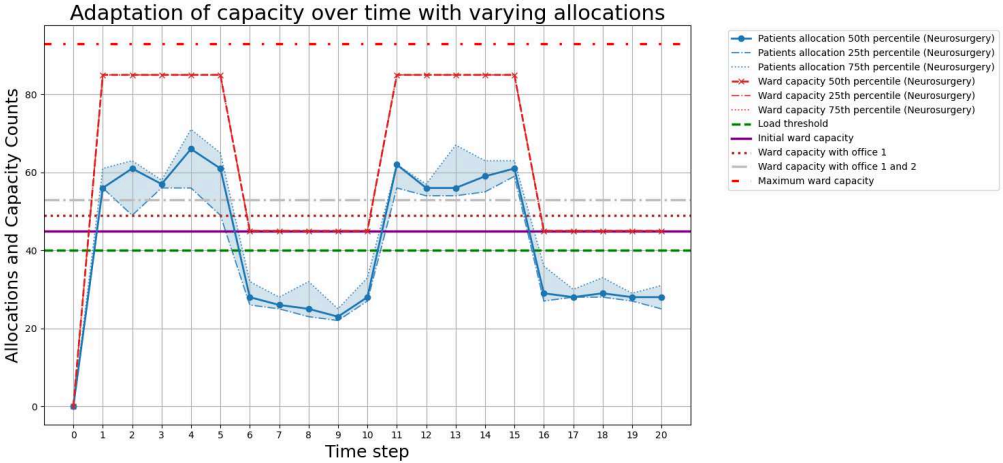
Allocation of patients



Allocation of patients



Allocation of patients (cont.)



Technical components

- The orchestrator and the lifecycle manager are both implemented through a **Backend API** to handle the requests

Technical components

- The orchestrator and the lifecycle manager are both implemented through a **Backend API** to handle the requests
- The dynamic information for the allocation are saved in a **PostgreSQL** database

Technical components

- The orchestrator and the lifecycle manager are both implemented through a **Backend API** to handle the requests
- The dynamic information for the allocation are saved in a **PostgreSQL** database
- The data of the treatments and hospital structure is retrieved by **SMOL** to create a runtime Knowledge-Base

Technical components

- The orchestrator and the lifecycle manager are both implemented through a **Backend API** to handle the requests
- The dynamic information for the allocation are saved in a **PostgreSQL** database
- The data of the treatments and hospital structure is retrieved by **SMOL** to create a runtime Knowledge-Base
- The patient needs are computed by **ABS** (an actor-based programming language)

Technical components

- The orchestrator and the lifecycle manager are both implemented through a **Backend API** to handle the requests
- The dynamic information for the allocation are saved in a **PostgreSQL** database
- The data of the treatments and hospital structure is retrieved by **SMOL** to create a runtime Knowledge-Base
- The patient needs are computed by **ABS** (an actor-based programming language)
- We solve the allocation by solving the constraints with **Z3**

Takeaways

- Runtime adaptation: opens/closes spare rooms in response to demand.

Takeaways

- Runtime adaptation: opens/closes spare rooms in response to demand.
- Alignment: semantic lifting keeps the DT consistent after human actions.

Takeaways

- Runtime adaptation: opens/closes spare rooms in response to demand.
- Alignment: semantic lifting keeps the DT consistent after human actions.
- Cost awareness: penalties encode policy (offices before corridor)

Takeaways

- Runtime adaptation: opens/closes spare rooms in response to demand.
- Alignment: semantic lifting keeps the DT consistent after human actions.
- Cost awareness: penalties encode policy (offices before corridor)
- Trade-off: correctness/alignment vs runtime overhead

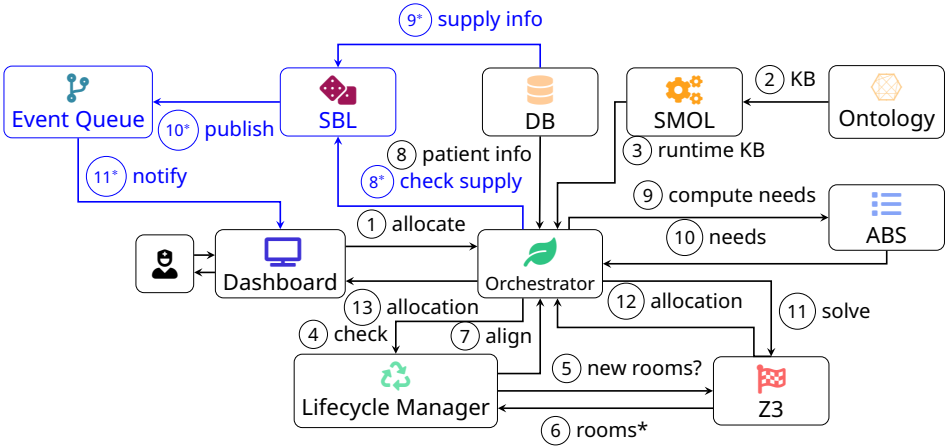
Takeaways

- Runtime adaptation: opens/closes spare rooms in response to demand.
- Alignment: semantic lifting keeps the DT consistent after human actions.
- Cost awareness: penalties encode policy (offices before corridor)
- Trade-off: correctness/alignment vs runtime overhead
- Current limitation: thresholds and penalties are manually tuned

Contents

1. Rationale
2. Digital Twin Concepts
3. Digital Twin Architecture
4. Extension to the DT

Extension to BedreFlyt with SBL



Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation

Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks

Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks
- The consumption and order is captured through a Markov Decision Process

Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks
- The consumption and order is captured through a Markov Decision Process
- The probability to transition from one state to the other are updated through Surprise-Based Learning

Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks
- The consumption and order is captured through a Markov Decision Process
- The probability to transition from one state to the other are updated through Surprise-Based Learning
- The update is triggered by checking for discrepancies between the observed behaviour and the expected one

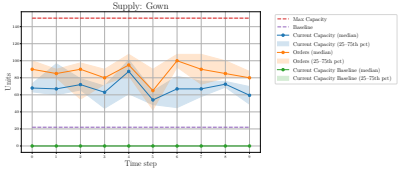
Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks
- The consumption and order is captured through a Markov Decision Process
- The probability to transition from one state to the other are updated through Surprise-Based Learning
- The update is triggered by checking for discrepancies between the observed behaviour and the expected one
- If supplies are required, an event is triggered

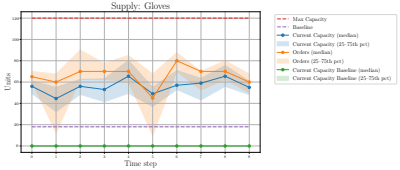
Extension to BedreFlyt with SBL (cont.)

- Apply supply management for allocation
- We consider the supplies required to complete the tasks
- The consumption and order is captured through a Markov Decision Process
- The probability to transition from one state to the other are updated through Surprise-Based Learning
- The update is triggered by checking for discrepancies between the observed behaviour and the expected one
- If supplies are required, an event is triggered
- The human-in-the-loop will then evaluate it and act accordingly

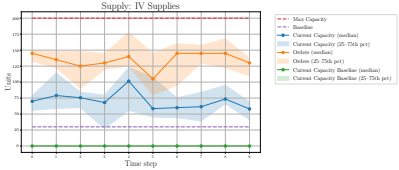
Results of the SBL approach



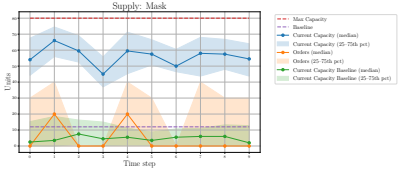
(a) Supply results with baseline for gowns



(b) Supply results with baseline for gloves

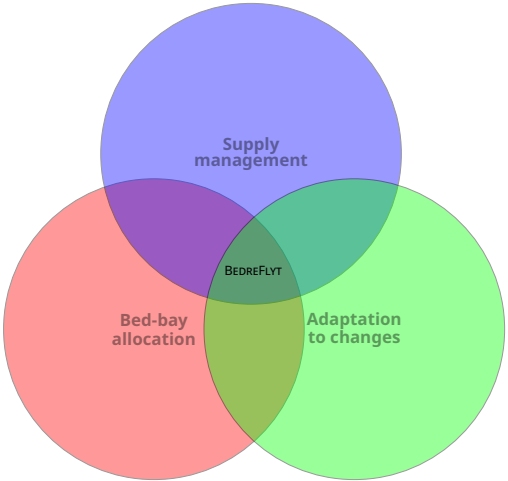


(c) Supply results with baseline for IV supplies



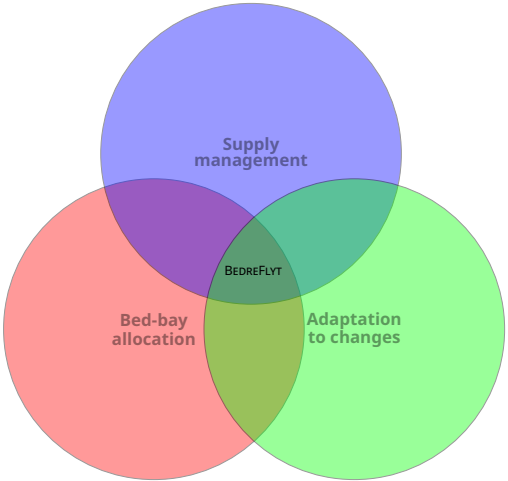
(d) Supply results with baseline for masks

Future directions



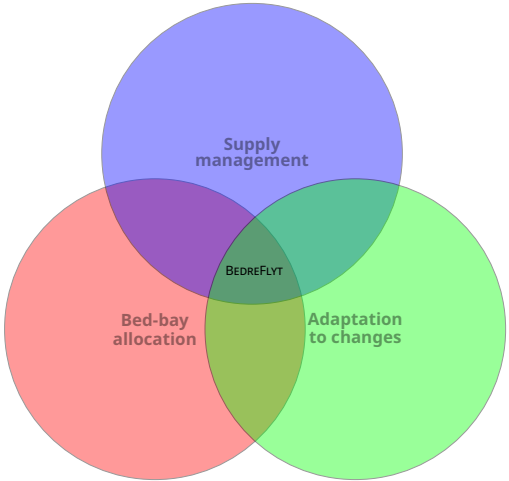
Future directions

- **Richer adaptation** (beyond load-only triggers, include constraint feasibility signals)



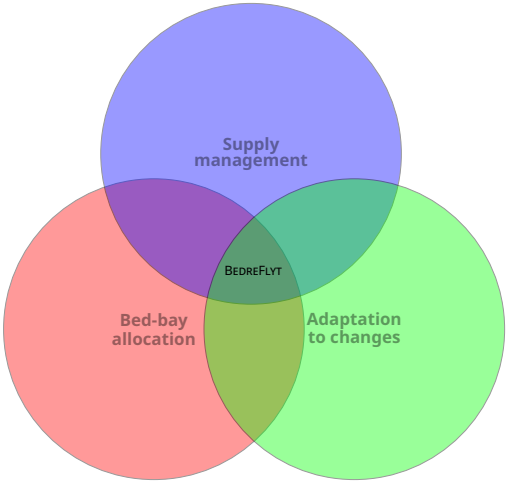
Future directions

- **Richer adaptation** (beyond load-only triggers, include constraint feasibility signals)
- **Learned penalties/thresholds** (data-driven tuning)



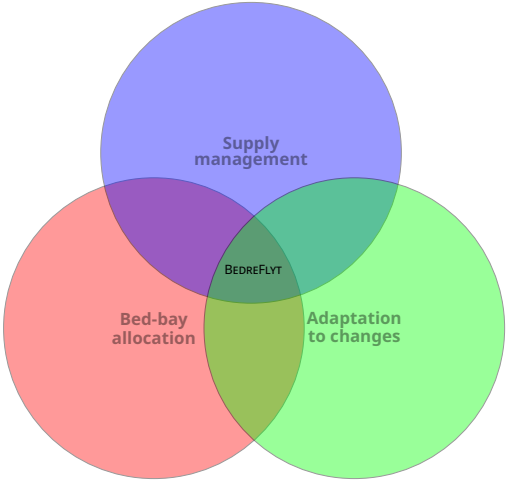
Future directions

- **Richer adaptation** (beyond load-only triggers, include constraint feasibility signals)
- **Learned penalties/thresholds** (data-driven tuning)
- **Live data integration** (connect to ward systems rather than simulator surrogate)



Future directions

- **Richer adaptation** (beyond load-only triggers, include constraint feasibility signals)
- **Learned penalties/thresholds** (data-driven tuning)
- **Live data integration** (connect to ward systems rather than simulator surrogate)
- **Broader scope** (multi-ward, staff constraints, other domains)



Thank you

Riccardo Sieve

riccasi@ifi.uio.no