A Self-Adaptive Digital Twin Architecture for Dynamic Resource Management

Riccardo Sieve
riccasi@ifi.uio.no
University of Oslo
Oslo, Norway

Paul Kobialka
paulkob@ifi.uio.no
University of Oslo
Oslo, Norway

Andrea Pferscher
andreapf@ifi.uio.no
University of Oslo
Oslo, Norway

Nelly Bencomo
nelly.bencomo@durham.ac.uk
Durham University
Durham, UK

Silvia Lizeth Tapia Tarifa
sltarifa@ifi.uio.no
University of Oslo
Oslo, Norway

Buster Salomon Rasmussen
buster.rasmussen@dscience.uio.no
University of Oslo
Oslo, Norway

Einar Broch Johnsen
einarj@ifi.uio.no
University of Oslo
Oslo, Norway

Abstract

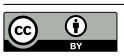
Digital Twins (DTs) are increasingly used to manage systems under fluctuating demand, yet many remain static and cannot adjust their internal models or control policies as the targeted real system changes. We present a self-adaptive DT architecture that supports runtime reconfiguration of resources. The architecture integrates *semantic reflection* to keep the DT's runtime model aligned with the structural representation of the real system, *lifecycle-based state management* to trigger reconfiguration at appropriate times, and *penalty-guided optimisation* for decision-making under constraints, balancing resilience and operational cost when capacity must be reorganised. We realise the approach in DynResDT, a resilient hospital ward prototype for bed bay allocation of patients. Through simulation with realistic patient-arrival patterns and stress peaks, the DT maintains model consistency, opens and closes overflow capacity when needed, and allocates patients while minimising costly room usage and unnecessary moves. Our results show a practical trade-off between correctness and responsiveness: timely and principled adaptions offset potential overhead through reflection and lifecycle logic. The architectural pattern is applicable beyond healthcare to other dynamic resource-management domains.

CCS Concepts

• **Software and its engineering** → **Layered systems**; *Software as a service orchestration system*; • **Computer systems organization** → **Self-organizing autonomic computing**.

Keywords

Digital Twins, Self-Adaptive Systems, Dynamic Resource Allocation, Bed Bay Allocation, Reflective Systems



SEAMS '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2445-9/2026/04
https://doi.org/10.1145/3788550.3794872

ACM Reference Format:
Riccardo Sieve, Paul Kobialka, Andrea Pferscher, Nelly Bencomo, Silvia Lizeth Tapia Tarifa, Buster Salomon Rasmussen, and Einar Broch Johnsen. 2026. A Self-Adaptive Digital Twin Architecture for Dynamic Resource Management. In *21st International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3788550.3794872

1 Introduction

Digital twins (DTs) are virtual representations of real systems, processes, or entities , designed to mirror and synchronise with their real-system counterparts [24] (if these systems are physical, the real system is often referred to as the *physical twin*, e.g., [11]). The digital twin's underlying models enable simulation, analysis, and optimisation of real-system behaviour over time. When bi-directional communication is established between the digital and real-system layers, the feedback loop is closed, allowing digital twins to not only reflect but also to influence, and possibly even control, the real system [23]. To ensure adaptation when the real system and its runtime and structural representation in the DT drift apart, semantic reflection, a technique to represent runtime states in knowledge graphs [20], enables the DT to reason about these differences and adjust autonomously. Self-adaptive systems have proven effective under changing conditions [35]. In safety-critical contexts, where reliability is paramount and failures can propagate, ensuring dependability strengthens overall robustness [37].

In domains such as healthcare, DTs have shown promise in improving operational decision-making, including patient flow and resource allocation [1, 2, 31]. In such dynamic domains, where patient influx, ward capacity, and seasonal trends vary rapidly, limited runtime adaptability remains a key barrier to effective resource management. Resource allocation tasks are still largely performed manually by admitting staff, who must make time-critical decisions, often under uncertainty and incomplete information. Manual task allocation in healthcare has shown to be inefficient and lead to ad hoc decision making [14]. When resources are insufficient, staff

resort to calling other wards, postponing treatments, or placing patients in makeshift areas such as corridors [38].

Decisions are further complicated by temporal dynamics in demand and capacity: patient conditions vary, resource availability fluctuates, and external factors such as seasonal illnesses (e.g., the annual flu) or extreme weather events introduce variability into demand patterns. Such continuous unpredictability makes static, deadline-based policies poorly suited to real operational settings [2].

To address this barrier, we propose a self-adaptive digital twin architecture, DynResDT, that evolves its internal models during operation. Our work integrates semantic models [13], simulation-driven forecasting [34], and constraint-based optimisation [4, 8] to enable runtime reconfiguration in response to changing system states. We scope the proposed architecture within BedreFlyt [22, 31], a DT for resource management which focuses on bed bay allocation in hospital wards. The system transforms a timed input stream reflecting patients arriving at the ward into a timed output stream of bed bay allocations for the ward. The DT design involves a human-in-the-loop to make the ultimate decisions on patient placements; thus, the actual bed bay allocation may differ from the allocation suggested by the digital twin. While our approach in this paper is designed to be generic and applicable in different domains, we scope our integration into bed bay allocation for validation.

Contributions and architectural focus. This paper presents a self-adaptive digital twin architecture for dynamic resource management. The contribution of this work lies not in any single mechanism in isolation, but in their integration into a coherent architectural framework that supports (i) runtime semantic alignment between the digital and physical twins through semantic lifting and reflection, (ii) controlled reconfiguration via lifecycle-based state management, and (iii) decision-making under competing objectives using penalty-guided optimisation, while explicitly supporting human-in-the-loop intervention. We demonstrate the feasibility of the proposed architecture through its instantiation in a hospital bed bay allocation scenario and an experimental evaluation of its adaptive behaviour in response to patient demands.

For validation, we consider the following research questions:
RQ1: *(Semantic reflection):* How can semantic reflection enable a DT to evolve its internal model at runtime to remain aligned with changes in the real system?
RQ2: *(Lifecycle adaptation):* How can lifecycle-based state management support timely reconfiguration of a DT under fluctuating resource demands?
RQ3: *(Decision-making under constraints):* How can penalty-guided optimisation enable trade-offs between resilience and operational cost during dynamic resource reconfiguration?

Concretely, we make the following contributions:

- A self-adaptive DT architecture that dynamically allocates resources under changing conditions;
- A runtime-evolving semantic model enabling model alignment via semantic reflection;
- A lifecycle manager implementing state-triggered reconfiguration through Optimization Modulo Theory-based optimisation;
- A prototype implementation and empirical validation on hospital ward bed bay allocation;

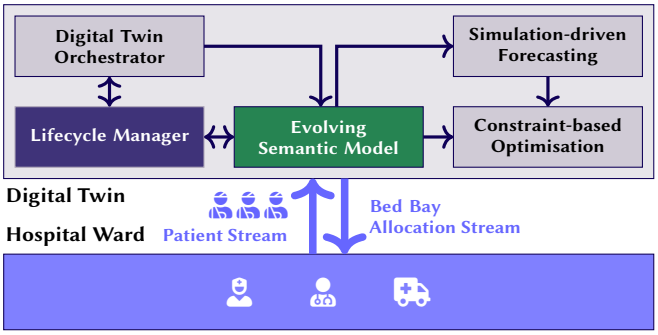


Figure 1: The self-adaptive DT architecture integrated in the BedreFlyt digital twin.

The paper is structured as follows: Section 2 introduces the BedreFlyt DT, motivating the proposed self-adaptive capabilities. Section 3 discusses related work on DT architectures, with a focus on self-adaptation and lifecycle management, and penalty-based optimisation. The proposed self-adaptive architecture is presented in Sect. 4, and its integration with BedreFlyt in Sect. 5. We evaluate DynResDT experimentally in Sect. 6, and discuss results and lessons learned in Sect. 7 before Sect. 8 concludes the paper.

2 BedreFlyt

BedreFlyt [22, 31] is a digital twin that aims to aid hospital staff with resource planning in hospitals by addressing the bed bay allocation problem. The real system that is twinned in BedreFlyt is a large hospital ward. The ward layout and its underlying components are modelled in the DT using a knowledge base that evolves over time. The DT uses this knowledge in combination with a timed input stream reflecting the patients arriving in the ward and their associated diagnosis. The DT is configured to provide meaningful suggestions for bed bay allocation in the ward. Since the knowledge on which the twin acts may be incomplete, the DT does not enforce the suggested bed bay allocation, but is part of a human-in-the-loop decision making process [39]. This means that that the bed bay allocation in the ward may differ from the one expected by the twin. The feedback loop is closed in the DT by dynamically adjusting the knowledge base based on the actual bed bay allocation as decided by the human, and then using the knowledge base to configure the models for the next analysis step.

Figure 1 illustrates BedreFlyt extended by self-adaptive capabilities. The existing DT integrates the following formal techniques into a tool chain for patient flow analysis: 1) a *Semantic Model* that formalises static knowledge about the structure and room layout of a hospital ward and domain knowledge about different diagnoses and their associated treatments, where each treatment details its resource needs, 2) a *Simulation-driven Forecasting* model to collect the resource needs of the patients in the ward at different points in time, and 3) a *Constraint-Based Optimisation* model to perform the actual bed bay allocation for each point in time. The *Digital Twin Orchestrator* uses semantic reflection [18, 19] to connect the *Semantic Model* to the other components (see Sect. 4.1 for details).

We now explain how the twin uses the semantic model. As shown in Fig. 1, BedreFlyt takes as input a timed *Patient Stream*, where

each patient has an associated diagnosis. The twin then queries the *Semantic Model* and stochastically selects a possible treatment for the diagnosis of each patient in the *Patient Stream* as input to the *Simulation-driven Forecasting* model, which in turn transforms this timed stream of patient data into a timed stream of resource needs that capture the bed bay allocation problems at different points in time. Together with a description of the ward's layout and capacities that is obtained from the *Semantic Model*, the timed stream of resource needs is turned into a stream of optimisation problems, which is given as input to the *Constraint-Based Optimisation* model. The result is a timed *Bed Bay Allocation Stream* of suggested solutions, returned to the human-in-the-loop decision-making process in the hospital ward.

As shown in Fig. 1, BedreFlyt transforms a timed patient arrival stream into a corresponding stream of bed bay allocation recommendations by combining a semantic representation of ward resources, simulation-driven forecasting, and constraint-based optimisation. The DT produces allocation suggestions that are reviewed and enacted by ward staff; the resulting allocation is synchronised back into the DT to maintain alignment between the virtual and real system states.

In this paper, our objective is to extend the digital twin architecture with self-adaptive capabilities to dynamically adapt the available resources of the twin, and hence the associated optimisation problem. We realise our solution as an extension of BedreFlyt. With respect to Fig. 1, the resources that were statically defined in the *Semantic Model* will now evolve over time by making additional rooms available in the ward and thereby increase or decrease the overall capacity of the ward when needed. To this aim, we introduce three new components in the DT architecture: (1) a *Lifecycle Manager*, (2) an *Evolving Semantic Model*, and (3) *Penalty-based Optimisation*. BedreFlyt acts as a digital twin of a specific hospital ward. The physical twin is the operational ward state (rooms, bed bays, active capacity) together with the observed patient flow. The virtual twin is an executable representation that combines the evolving semantic model (knowledge graph), simulation-based forecasting, and optimisation. Semantic lifting and alignment synchronise observed changes into the runtime model, and the architecture closes the loop by producing reconfiguration recommendations (enacted by a human operator in our setting). In our evaluation, the existing simulator serves as a surrogate source of observations, while the architecture is designed to connect to live ward data in deployment.

3 Related Work

This paper presents a self-adaptive DT architecture to optimise resource management. Different self-adaptive architectures for DTs have been proposed, using, e.g., MAPE-K loops for adaptive robot control [9] and cyber-physical system controllers [10], quality awareness to capture drifts in real system behaviour [32], combinations of system engineering techniques and information technology to adapt the DT lifecycle [26], leveraging semantic technology [16, 19]. In contrast to adaptation based on quality awareness [32], our system is driven by penalties associated with the usage of additional resources, which affects the associated optimisation problems. Compared to these proposals, our architecture integrates semantic reflection for runtime model alignment, lifecycle-based state management for structured reconfiguration, and penalty-guided optimisation for decision-making under constraints.

Within the domain of DTs, semantic reflection has proven a powerful tool to adapt the runtime state of the heap [18]. This process is achieved using reasoners to identify drifts in the runtime knowledge graphs to adapt the underlying model [19] while ensuring correctness of the constructed DT [16]. Semantic reflection has also proven effective for runtime adaptation of the model, both for structural drifts in the runtime states [21], as well as behavioural drifts [29] (when the component is not mutated, but its information requires it to change its behavioural state).

While Sieve et al. [29] showed how to integrate behavioural self-adaptation using SMOL and semantic reflection, other approaches have been proposed to achieve self-adaptation of DT behaviour. In particular, declarative lifecycle management [15] separates a model of the different states of the lifecycle from the conditions that trigger the adaptation. Furthermore, lifecycle management has proven effective, when in conjunction with real systems replicas, to improve the scalability of the development of the DT, improving modularity [25]. Lifecycle management in DTs, when conjoined with machine learning approaches, has proven powerful to handle complex scenarios and improve management, suggesting potential advantages of combining the two technologies [28].

Optimisation Modulo Theories (OMT) [4] adds an objective function to a satisfiability problem. However, it can be challenging to integrate different requirements with the objective function. To this aim, a set of (static or dynamic) penalties can be integrated into the constraint problem [36]. When dealing with different objectives, one set of weights might not be enough, and using functions with specific weight vectors can allow the system to adapt the optimisation problem with different requirements [27]. While this paper uses static penalties defined in the semantic model, we can use semantic lifting to adapt them at runtime and make them dynamic. Tessema et al. [33] proposed a self-adaptive penalty function to solve a constrained optimisation problem, using distance value and penalties to identify the best infeasible individuals in a population. This approach would be interesting for future work.

Although these works have proven powerful in self-adaptation and optimisation for constraint solvers, the integration of semantic reflection, lifecycle adaptation, and penalty-guided optimisation into a single runtime architecture has, to the best of our knowledge, not yet been explored. Our work addresses this gap by integrating the different aspects into a runtime self-adapting DT architecture.

4 A Self-Adaptive Digital Twin Architecture

In this section, we propose a self-adaptive DT architecture that improves resource management in an evolving target system. Through periodic monitoring of the target system's operational state, we trigger the self-adaptive capabilities of the DT that adapt the runtime states of the DT in response to internal and external changes.

Boundary and assumptions. In this paper, the *digital twin* comprises the BedreFlyt-based software system, including its evolving semantic model, orchestration logic, lifecycle management, and optimisation components. The *physical twin* is the hospital ward (beds/bays/rooms) and its operational state, with changes enacted by staff (e.g., opening or closing rooms and placing patients). The

environment includes the patient arrival stream and operational policies, while *operators* (ward staff) act as *human-in-the-loop* decision makers whose actions may diverge from DT recommendations and are subsequently reflected back into the DT via the synchronisation mechanisms described in this section. Human-in-the-loop decision making is a well-established assumption in self-adaptive systems, particularly in safety-critical domains, where automated adaptations provide recommendations that may be overridden by human operators [6].

The requirements for the self-adaptive DT architecture must include support for dynamic resource management in a consistent and accurate manner, to accommodate different scenarios when managing resources in the target system. To accommodate these requirements, we designed our architecture by integrating three main components: the *Evolving Semantic Model*, the *Digital Twin Orchestrator*, and the *Lifecycle Manager*. A purpose of the *Evolving Semantic Model* is to maintain a knowledge graph that represents the structure and relationships of the resources in the target system (thus, the *Semantic Model* of Sect. 2 here represents one specific instance). The *Evolving Semantic Model* will further incorporate penalties for spare resources, which can be utilised during optimisation to manage resource allocation effectively. The evolving semantic model acts as a runtime model of the target system, supporting semantic reflection and adaptation during operation [3, 5].

Without semantic lifting and subsequent reflection, deviations introduced by operator decisions or emergency practices would remain invisible to the DT's reasoning mechanisms, leading to model drift and potentially unsafe optimisation decisions. The choice of such a model enables integration with reasoners to support advanced decision making. The *Digital Twin Orchestrator* coordinates component interactions and manages the adaptation process. The *Lifecycle Manager* monitors DT component states, triggers adaptations under predefined conditions, and applies penalty-based optimisation to determine resource allocations, enabling dynamic adaptation while minimising associated penalties.

Key design challenges include integrating components, detecting and responding to lifecycle state changes, and maintaining the consistency of the *Evolving Semantic Model* during runtime adaptation to preserve alignment with the real system.

The remainder of this section describes semantic model adaptation via the *Digital Twin Orchestrator* (Sect. 4.1), followed by lifecycle management (Sect. 4.2) for monitoring states and triggering adaptation when changes are detected.

4.1 An Evolving Semantic Model

The *Evolving Semantic Model* forms the basis for self-adaptation in the DT. It is a knowledge graph that virtually represents the data and relations regarding the structure of the different resource concepts of the target system, as well as domain knowledge concerning their use. The *Evolving Semantic Model* captures resource concepts with attributes that capture which resources are in a system by default and which resources are *spare*, i.e., resources that are not available by default, but can be activated by the target system if required. In the *Evolving Semantic Model*, spare resources have associated penalties expressing the cost of making these resources

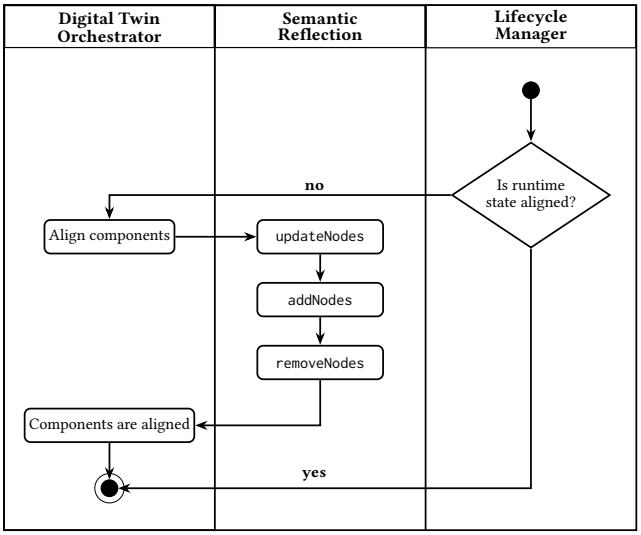


Figure 2: Alignment procedure between the runtime knowledge graph of the DT and the knowledge graph in the *Evolving Semantic Model*.

available. These penalties are used by the *Lifecycle Manager*'s constraint solver to optimise penalty costs when solving the resource configuration allocation problem, e.g., to generate a configuration that has sufficient capacity, with minimum penalty.

Drifts in the underlying domain model can happen both from changes in the overall behaviour (e.g., the system needs to adapt due to a lack of resources) or because of structural changes (e.g., the user might add, update or remove one or more components in the real system). To keep track of these cases and ensure that we can properly align the *Evolving Semantic Model* with the real system, we make use of *semantic lifting* and *semantic reflection* to enrich the knowledge graph in the *Evolving Semantic Model*. Semantic lifting [18] is a technique to create an additional knowledge graph of the runtime state of the DT. Semantic reflection [17, 20] then combines the knowledge graph of the runtime state with the *Evolving Semantic Model* for introspection and reasoning. Our architecture makes use of semantic reflection to reason in the knowledge graph of the *Evolving Semantic Model* about changes in the configuration of available resources in the real system. Semantic lifting here ensures a consistent semantic model throughout the overall execution. In our implementation, semantic reflection is realised using SMOL [18, 19], a programming language that supports the lifting of runtime states into a knowledge base, which can be accessed in programs via SPARQL and SHACL queries (e.g., [12]).

4.2 Lifecycle Management

The *Lifecycle Manager* organises the self-adaptive capabilities for resource management in the DT. We capture the condition associated to the different states of a lifecycle as concepts in the *Evolving Semantic Model*, and we make use of penalty-based optimisation when changing the configuration of available resources in the *Semantic Model*, depending on the state in the lifecycle of the monitored

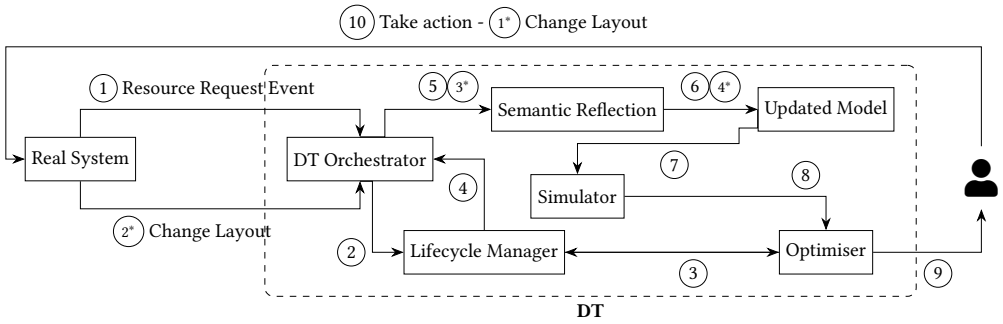


Figure 3: Operation flow for resource management and adaptation for changes in the layout (identified by *).

system. Declarative lifecycles [15] separate the conditions for self-adaptation (the "*when*"), here partially formalised in the *Evolving Semantic Model*, from the way the self-adaptation is realised (the "*how*"), here implemented in the *Digital Twin Orchestrator*.

The *Lifecycle Manager* monitors the different components of the real system with respect to conditions in the current state of their lifecycles. If these conditions indicate a change to a different state in their lifecycle, the *Lifecycle Manager* triggers a self-adaptation process in the DT through semantic reflection. Namely, for each resource component that is considered during the self-adaptation process, we use their associated penalties to ensure that that the system reconfigures to an optimal configuration of resources that minimises the incurring penalty. This way, the lifecycle management process leverages penalty-based optimisation in collecting and analysing penalties associated with the different resources that can be added to the system, and minimises the overall penalty from adding the external resource to the resource management process.

The *Lifecycle Manager* is implemented as a separate component, invoked periodically by the *Digital Twin Orchestrator* to find a new optimal system configuration. Keeping the *Lifecycle Manager* as a separate component enhances modularity and separation of concerns in the DT architecture and allows for easier maintenance and potential future extensions, such as adding more complex lifecycle management strategies or integrating with other systems. Figure 2 illustrates how the *Lifecycle Manager* aligns with the *Evolving Semantic Model* and the *Semantic Reflection Layer* via the *Digital Twin Orchestrator*. The *Lifecycle Manager* monitors the runtime state of the DT, e.g., the current state in terms of current resource usage and incoming resource demand, and based on the penalties associated with the different resource components, it updates the *Evolving Semantic Model* with a new optimal configuration of resources by prompting the *Digital Twin Orchestrator*, when needed. The *Digital Twin Orchestrator* then aligns the new configuration of resources to the DT's knowledge graph (see Fig. 2), considering both the structural and runtime knowledge graphs in the *Evolving Semantic Model* of the self-adaptive DT. The alignment uses the following operations, all based on semantic reflection:

(1) `updateNodes` checks whether the runtime state needs to be updated by aligning the attributes that might have changed.
(2) `addNodes` checks for nodes that have to be added in the runtime knowledge graph and adds them, if needed.
(3) `removeNodes` checks for nodes that have to be removed in the runtime knowledge graph and removes them, if needed.

While we aligned existing nodes first and removed former nodes last, this order of the operations is not binding. The operational flow of the DT is shown in Fig. 3, detailing how the DT components interact to address both resource requests and reconfiguration requests.

BedreFlyt processes the operational stream in discrete time steps. At each step, new patient events may create a resource-request condition. The Digital Twin Orchestrator invokes the Lifecycle Manager once per step (i.e., periodically) to evaluate thresholds and, when needed, initiate reconfiguration via the Optimiser and Semantic Reflection. The result is then returned to the human operator, who will make the final decision and take action. Since the human operator may make changes to the suggested configuration of resources, the DT will update the model, using *semantic lifting* to represent the new configuration.

The lifecycle management process can be tailored to specific needs, with respect to when the process should be triggered. In our implementation, we let the *Lifecycle Manager* be triggered periodically, using a simple notion of time steps to capture the passing of time. However, it is straightforward to adapt the architecture to use conditional triggers instead.

By integrating penalties associated with resource usage into the optimisation model, the DT can make more informed decisions that balance resource usage with operational costs. The penalty-based optimisation is integrated into the existing *Constraint-based Optimisation* model of BedreFlyt, explained below. The penalties improve the optimisation model by accounting for the costs associated with using spare resources when determining the optimal configuration of resources and how they should be used.

5 DynResDT

We now discuss the integration of our self-adaptive DT architecture with the BedreFlyt DT. Previously, BedreFlyt [22, 31] assumed a fixed layout of a hospital ward in the *Semantic Model*. Our approach lifts this assumption to allow different ward layouts, increasing or decreasing capacity by, respectively, opening and closing rooms for bed bay allocation. This scenario shows how a DT can adapt to a stressful workload caused by incoming patients by changing to a different ward layout depending on the workload of the ward.

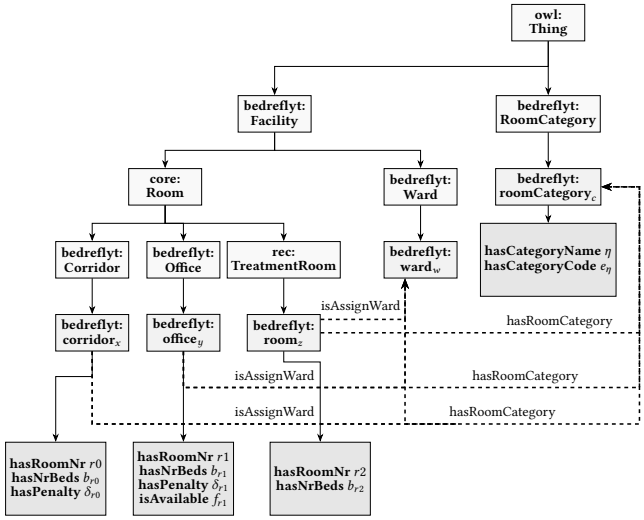


Figure 4: Part of the ontology in the Evolving Semantic Model that captures the ward layout.

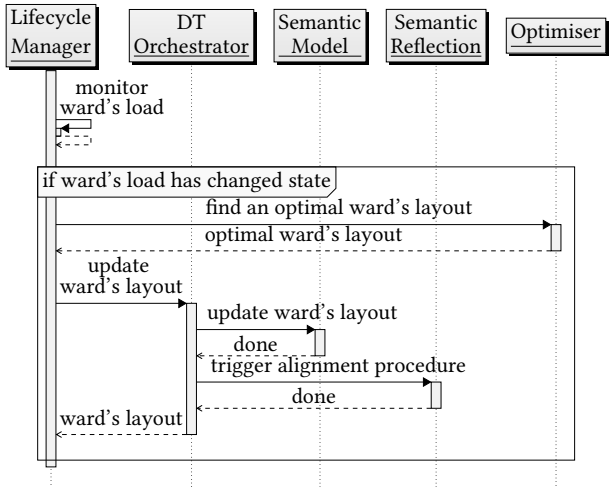


Figure 5: Lifecycle manager procedure for DynResDT.

The Evolving Semantic Model. In BedreFlyt, the existing layout of a hospital ward is captured in its *Semantic Model* [22, 31]. The ward is composed of a set of treatment rooms, each with a certain capacity (number of bed bays) and category, e.g., *High Monitoring*. We extend the existing *Semantic Model* of the ward with spare rooms, which corresponds to corridors and offices that can be used to temporarily accommodate patients. Furthermore, such spare rooms have an associated penalty greater than zero, while treatment rooms that are in the default layout have no penalty. Figure 4 shows part of the ontology of the *Evolving Semantic model*, which captures the layout of hospital wards with extra spare rooms concepts and their associated penalties. The *Digital Twin Orchestrator* lifts the layout of the ward in the ontology during the alignment procedure. The *Evolving Semantic Model* also includes the parameter ε that allows to monitor the lifecycle conditions of the ward. Concretely, we set the threshold ε, which indicates the percentage of capacity at which the ward is considered to be under- or over-loaded. The integration of the *Evolving Semantic Model* in the existing BedreFlyt architecture allows the DT to adapt resource supply in response to patient demand.

The Lifecycle Manager. We consider a *Lifecycle Manager* for the load of the ward. Given the initial default capacity of a ward Ω, the load threshold Γ is a number calculated as follows: $\Gamma = \lfloor \frac{\Omega \cdot \varepsilon}{100} \rfloor$, with threshold ε from the *Evolving Semantic Model*. Given a current number of occupied bed bays Θ, to calculate the conditions in each state of the lifecycle, the *Lifecycle Manager* uses the following states: (1) *under-loaded* if $0 \leq \Theta \leq \Gamma$ and (2) *over-loaded* if $\Theta > \Gamma$. The periodic triggers on the *Lifecycle Manager* aligns both the layout as shown in Fig. 5, and the Γ when the capacity changes.

Extending the Constraint-based Optimisation Model. The existing *Constraint-based Optimisation* model of BedreFlyt solves the bed bay allocation of patients, detailed in [22, 31]. It encodes an optimisation problem that considers constraints related to room capacity, gender of patients, isolated patients, patients' monitoring needs (related to the room categories), and minimisation of room reallocation of patients during their hospitalisation period. In this paper, we further extend the *Constraint-based Optimisation* model to solve the ward layout and refine the solution for the bed bay allocation of patients to include the penalties associated with rooms.

We encode an optimisation problem that finds an optimal ward layout for a given set of patients P. Let R' be a collection of rooms that can be opened in a given ward. For room $r \in R'$, $o_r \in \{0, 1\}$ indicates whether r is opened, $o_r = 1$, or closed, $o_r = 0$. Let $\delta_r \in \mathbb{N}$ be the penalty associated with r and $b_r \in \mathbb{N}$ the capacity of r. The objective function to minimise the maximum penalty for opening rooms required to host P patients is

$$\min \quad \sum_{r \in R'} o_r \cdot \delta_r \cdot b_r,$$

subject to

$$\sum_{r \in R'} o_r \cdot b_r \geq |P|$$

with the constraint of uniqueness for the different $\delta_r \cdot b_r$. To find an optimal assignment for the bed bay allocation of patients, in addition to the existing constraints, we also consider penalty minimisation in the equation below, which accumulates penalties for each patient's room assignment. To encode this constraint, we introduce the following variables. For patient $id \in P$ and room $r \in R$, variable $a_{id,r} \in \{0, 1\}$ encodes whether patient id is placed in room r, and σ_{id} encodes whether the patient was moved from their previous bed bay. We now express the minimisation of penalties for the bed bay allocation problem as follows:

$$\min \sum_{id \in P} \sigma_{id} + \sum_{id \in P} \sum_{r \in R} \delta_r \cdot a_{id,r}$$

The *Constraint-based Optimisation* model uses Optimisation Modulo Theories (OMT) and the tool Z3 [7] to solve both optimisation problems. The solver finds optimal solutions for both (1) the ward layout and (2) the bed bay allocation with minimal penalties.

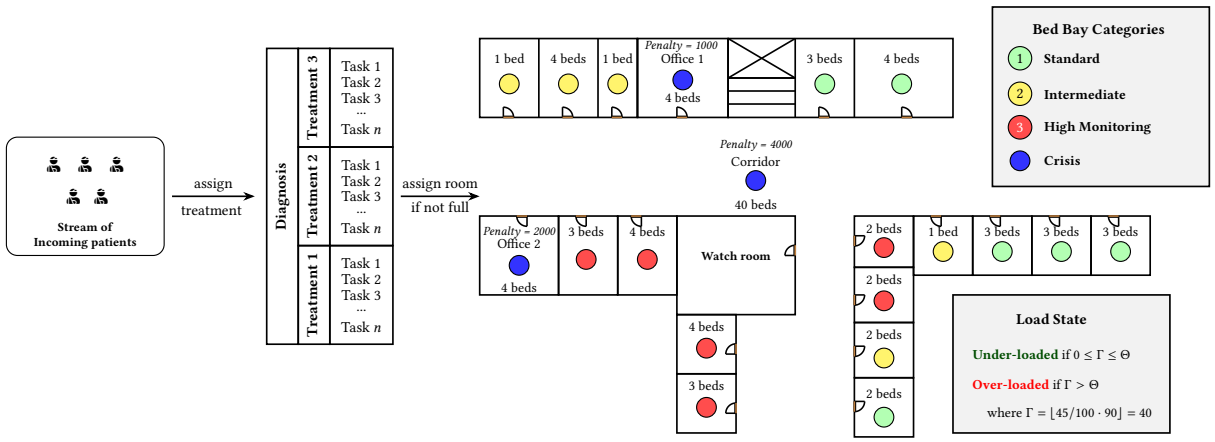


Figure 6: Hospital ward scenario used to evaluate DynResDT, specifying room capacities/categories, spare-room penalties, and the ward-load lifecycle threshold. Here Θ is the number of occupied beds and Γ is computed as in Section 5.

6 Evaluation

In this section, we address *RQ1–RQ3* by evaluating DynResDT under realistic, varying-load scenarios[1]

6.1 Experimental Setup

For evaluation purposes, we use the ward in Fig. 6 with load threshold $\varepsilon = 90\%$ in the *Evolving Semantic Model*. The ward has three room categories (*Standard*, *Intermediate*, *High Monitoring*) plus *Crisis* spare rooms. Default capacity is 45 bed bays and maximum capacity is 93, enabled by 48 temporal resources progressively activated when $\Gamma = 40$ (i.e., 90% of default capacity) is reached. The *Crisis* category simplifies spare-room identification and allows allocation to any patient type, subject to contagion constraints.

Events. To simulate the patient influx, we developed an *Event Stream Generator* (ESG) that produces different allocation scenarios. The ESG samples the number of patients arriving in a day from a Poisson distribution with average λ. The construction of individual events is modular, splitting the steps for generating an event by sampling of different distributions. In our experiments, the 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.

Event Orchestration Engine. The *Event Orchestration Engine* (EOE) triggers the sampling of ETG and ESS, and appends the generated events to the *Future Events Queue* (FEQ). The EOE ensures that at least one event is always stored in the FEQ, and that the events in the FEQ are sorted in ascending order by their timestamps.

When running the EOE, events are generated in the 7-step process, depicted in Fig. 7. The process is detailed as follows:

(1) Initialise the FEQ with an initial event
(2) Loop through the event generation procedure until the desired number of steps is generated

[1]The results reported in this section can be reproduced with the additional material [30].

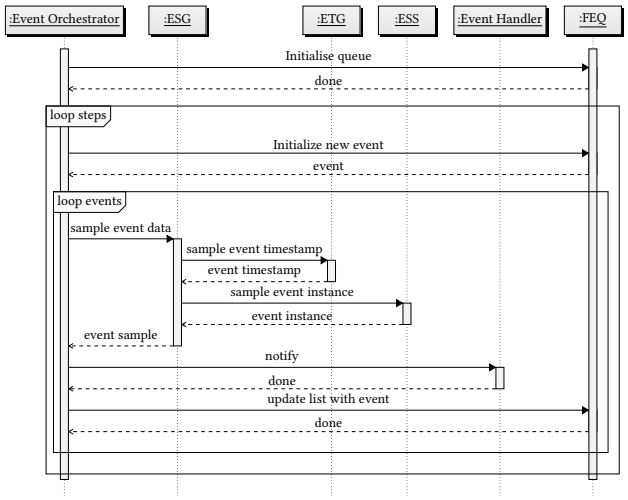


Figure 7: ESG procedure for Experiments.

(3) Loop through each step and initialise the events by sampling via the ETG and ESS component
(4) Notify the event handler of the new event
(5) Append the event to the FEQ

Peak generation. To further enhance the capability of the ESG, we extend it 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 every ν time steps, and increasing and lowering the occurrences by a factor of ξ.

The expectation of the Poisson distribution is chosen based on realistic-world data of incoming patients, see Fig. 8, and with actual diagnoses and treatments performed in the ward.

The ESG generates a stream containing, for each time step, a patient identifier and a corresponding treatment. Concretely, (1) to answer *RQ1*, we test the system with the ESG with an average λ of 40 for the Poisson distribution, to vary the incoming number

Table 1: Execution times of the simulation for the different elements that compose DynResDT.

Component	50^{th} percentile (s)	Average (s) [*Std. dev. (s)*]	Max Value (s)	Min Value (s)
Data Retrieval	0.0090	0.0108 [0.0103]	0.1310	0.0040
Optimal Ward Layout Allocation	0.0000	0.0019 [0.0035]	0.0290	0.0000
SMOL Alignment	29.6330	286.8063 [521.0410]	3917.1150	0.0000
Components Retrieval	9.5280	51.2426 [75.5458]	278.7290	1.1140
Simulation-driven Forecasting	1.2360	1.5462 [3.5318]	47.8900	0.9270
Bed Bay Allocation	0.1290	0.1289 [0.0258]	0.2110	0.0720
Overall Execution (s)	40.535	339.7368 [600.1582]	4244.1050	2.1530

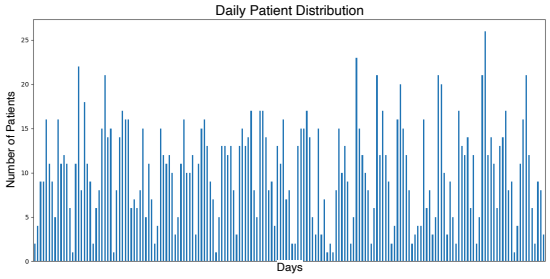


Figure 8: Distribution of patients in a 6 months span of time. The graph shows the number of new incoming patients on a daily basis; the state of the ward is not captured since some treatments require a multi-day stay at the hospital.

sufficiently, autonomously triggering the semantic reflection alignment process during the adaptation of the capacity of the ward. By varying the number of patients frequently, the opening and closing of additional rooms is demonstrated, as well as the different selections of room combinations; (2) to answer *RQ2*, we use the ESG with peak generation. Thereby, the expected number of incoming patients each day fluctuates between ≈ 30 and ≈ 60 every five steps.

The experiment demonstrates the principled adaptation capability of the architectural design to properly adapt to unpredictable scenarios; for *RQ3*, we show how penalty-guided optimisation enables cost-aware decisions under dynamic demand, preserving resilience. Specifically, the *Lifecycle Manager* selects additional rooms by minimising cumulative room penalties; thus, additional rooms are only opened when the additional capacity is needed.

6.2 Results

RQ1: Semantic Reflection. For the first research question, we want to evaluate whether the *Lifecycle manager* triggers self-adaptation when needed via the procedure depicted in Fig. 5. The *Lifecycle manager* further calls the alignment procedure with semantic reflection, depicted in Fig. 2. This is done to ensure that the *Evolving Semantic Model* and the knowledge graph with the runtime state of the DT are aligned and adapt according to the current ward's layout. This property is critical in DT architectures, as it must remain trustworthy under structural changes, not only in the healthcare domain but across dynamic resource management domains in general.

As shown in Fig. 9, when the number of incoming patients were going above the load threshold $\Gamma = 40$, extra room were pre-emptively given to the system based on the requirements and. With the set of penalty chosen, the offices took precedence for opening and, when the number of patients was too high for a single office, or both offices, the corridor was then opened to supplement the system with extra capacity. Furthermore, to avoid moving patients around rooms, as it is the primary goal of the allocation process, spare rooms are closed only when the load goes below the Γ. While, getting below a certain threshold might make the system more reactive by switching from the corridor to the offices, it would move patients, violating the main objective.

We further define the following research sub-question

RQ1.1. (integration & scalability): How does the integration of semantic reflection, lifecycle management, and optimisation affect scalability and responsiveness ina self-adaptive DT?

To address *RQ1.1*, we computed the execution time of the different components of DynResDT and, for each, the median, average, and standard deviation of the different times, the execution time of these different components that are recorded during the running scenario is shown in Tbl. 1, where:

- *Data Retrieval* reflects the execution time of retrieving the current allocation of patients and the current ward layout.
- *Optimal Ward Layout Allocation* reflects the execution time of the OMT problem that find the subset of available rooms to open/close that minimises the overall penalty.
- *SMOL Alignment* reflects the execution time that processes the semantic reflection procedure.
- *Components Retrieval* reflects the time to process the different elements required for the bed bay allocation process. The execution time can fluctuate during the adaptation process since elements change and the cache needs to be updated.
- *Simulation-driven Forecasting* reflects the execution time of the simulation model to collect the patient needs.
- *Bed Bay Allocation* reflects the execution time for the bed bay allocation problem to solve the OMT problem for the allocation of patients in the ward.

The different values show good performance for most of the components, with a higher overhead for the semantic reflection process that is required by SMOL to achieve alignment. While the overhead is significant, the median for the component is still within a reasonable execution time.

RQ2: Lifecycle adaptation. To ensure the correctness of the lifecycle manager approach, we need to ensure that the system can correctly reflect the two states in the lifecycle of the load in the

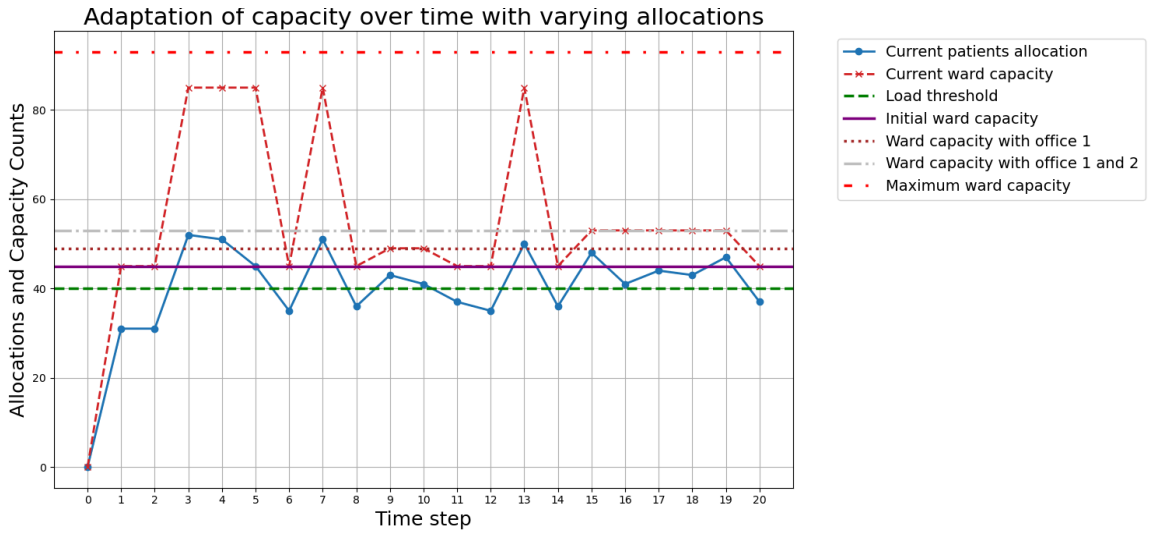


Figure 9: Single-iteration scenario showing ward capacity adaptation under varying patient load. Capacity increases incrementally by opening appropriate rooms as demand changes. Maximum capacity includes offices and the corridor. To reduce movement, adaptation is skipped when demand can be met by opening a single office (e.g., time step 5).

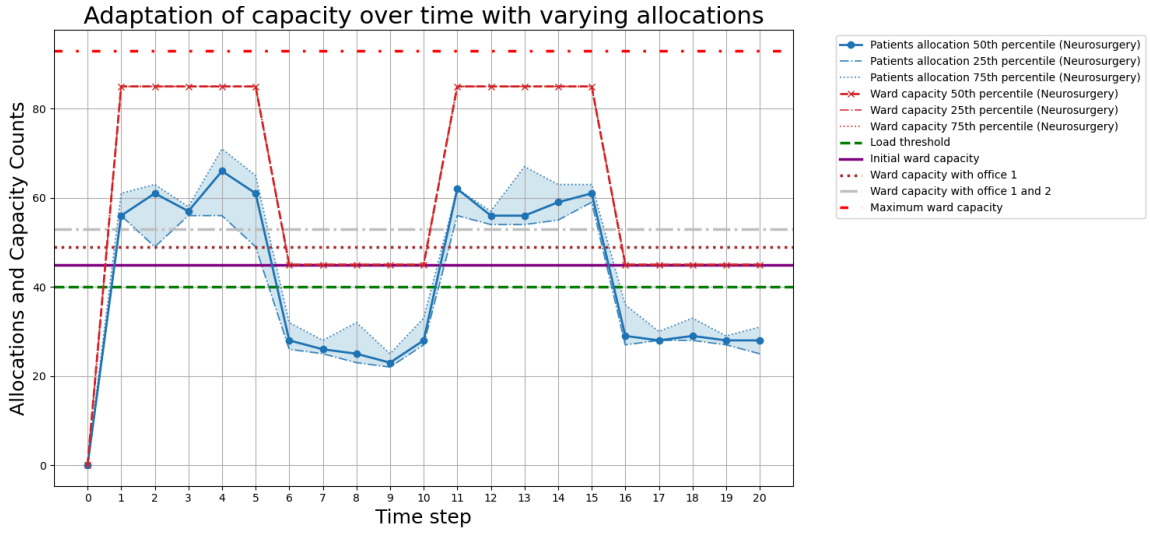


Figure 10: Combined running scenarios that show the variation of ward capacities with a varying number of patients over ten different iterations. The distribution has been evaluated by creating peaks in the number of incoming patients to mimic the condition of high stress and crisis in the hospital ward. The different range of values defined by the different percentiles is also shown to describe the variation that is given by the ESG.

ward, namely *Under-loaded* and *Over-loaded*, at all times, especially when the ward is exposed to variation of patients with peaks, requiring the DT to avoid both premature reconfigurations and delayed responses. While the operational results highlighted in *RQ1* showed timely allocation of spare rooms, the lifecycle manager provides a principled temporal structure for adaptation.

When the number of patients varies heavily over time, as in a crisis scenario where the hospital is expected to be stressed and overloaded, the adaptation process is critical to ensure that the hospital is resilient and can maintain operations for a prolonged period of time, as well as to being able to get back to normal operational conditions after the crisis is over. To simulate a crisis scenario with sudden patient peaks, the experiment is using the peak generation feature of the ESG introduced in Sect. 6.1 by applying the process with time steps $\nu = 5$ and factor $\xi = 1.5$ that increases and lowers the occurrences to ensure that, for non-peak time, the ward resource needs could be covered without spare rooms, and, for periods of time with high load, a default capacity layout will not be enough to accommodate the incoming patients, and the ward needs to increase its capacity. The overall process is then achieved and

Table 2: Execution times over ten iterations for DynResDT components under high-load patient peaks.

Component	50^{th} percentile (s)	Average (s) [*Std. dev. (s)*]	Max Value (s)	Min Value (s)
Data Retrieval	0.0090	0.0114 [0.0132]	0.1460	0.0050
Optimal Ward Layout Allocation	0.0000	0.0021 [0.0157]	0.2090	0.0000
SMOL Alignment	0.0020	21.3316 [52.9186]	474.1070	0.0000
Components Retrieval	2.8700	7.3759 [9.9360]	44.6160	0.7480
Simulation-driven Forecasting	1.2340	1.2742 [0.2218]	2.5610	0.8900
Bed Bay Allocation	0.1225	0.1364 [0.0614]	0.4170	0.0440
Overall Execution (s)	4.2375	30.1316 [63.1667]	522.0560	2.1280

shown in Fig. 10, where experiments show that the self-adaptive system was consistently able to respond to the crisis period and ensure allocation of incoming patients.

This illustrates that the *Lifecycle Manager* described in Sect. 4.2 can serve as a reusable design pattern for resource management in DTs. Experiments confirm that in our evaluation scenario, whenever the load fluctuates over time, and the monitored component (e.g. the ward load) changes state according to certain conditions, lifecycle can declaratively capture the monitoring conditions attached to each state and trigger a self-adaptation process.

Our experiments in Fig. 10 show that when the patient influx is more consistent over time (i.e., the current capacity in the ward is not fluctuating as often as in the experiments shown in Fig. 9), the overhead of the adaptation process is lower due to better stability of spare rooms for longer periods. This improvement is reflected in Tbl. 2, where, compared to Tbl. 1, the semantic reflection process introduces less overhead, since there are fewer fluctuations in the load, which triggers fewer requirements for the system to adapt.

One further aspect that might affect the adaptation procedure in the *Lifecycle Manager* is the load percentage threshold ε, stored in the *Evolving Semantic Model*, that is used to calculate the load threshold Γ that is further used to determine the conditions of the states in the lifecycle of the ward load. If the threshold is too high, the system might not open spare rooms before it exceeds its maximum capacity, requiring extra effort in the immediate time; if it is too low, the system might open the spare rooms too soon, resulting in a non-needed computational overhead. As such, it is important to find the right ε that balances between the two extremes. This, at the moment, is done ad hoc in our system; it could be interesting to automate the process of finding an optimal value for ε; however, this, for now, remains as future work.

RQ3: Decision-making under constraints. Results from the previous questions highlight the resilience of DynResDT to deal with uncertainties in the resource needs. To answer *RQ3* and evaluate trade-offs between resilience and penalty cost, we now focus on penalties. In our evaluation scenario, we selected a set of penalties to be incurred by the system if the corridor or spare rooms are used when applying the penalty-related constraints (see Sect. 5).

In our scenario described in Fig. 6, we modelled the ward with lower penalties for office usage, and higher penalties for corridor usage. This choice reflects conditions in which patients might be contagious, and isolation would be the choice for adaptation. To avoid using spare rooms when there is no need, and to always prefer offices over the corridor, we set the penalties to be 1000 for Office 1 and 2000 for Office 2, and the penalty for the corridor to be 4000. Our experiments in Fig. 9 showed that such wanted behaviour was respected by the self-adaptive procedure, where Office 1 was always chosen when the extra requirement was enough to be covered by a single spare office; even if the capacity of both offices is the same, the first one was always chosen due to the incurring minimum penalty calculation. The choice of penalties and the previous experiments in *RQ1* and *RQ2* show that, if there is a solution for the allocation of patients to bed bays, DynResDT will find such a solution, and it will be minimal with respect to the incurring penalties, showing the trade-offs between resilience and cost.

While the chosen penalties ensured that a solution was always found with the desired layout, the choice of penalties is in this work fairly arbitrary, and set to be sufficiently high and distributed for the penalty-based optimisation to always have a consistent behaviour with respect to a given policy. Remark that a strategy for penalties could also change over time, to reflect, e.g., seasonality. Since penalties are captured in the *Evolving Semantic Model*, the architecture makes it easy to manually change them to reflect other policies. It would be interesting to automate the process of finding optimal values for penalties, but this remains future work.

Summary of Results. *RQ1* was answered by reporting that the adaptation process occurred correctly with fluctuations of incoming patients exceeding or dropping below the lifecycle load threshold $\Gamma = 40$, shown in Fig. 9. *RQ2* was answered by Fig. 10, where the variation of patients with peaks in the allocation was correctly captured. Table 2 shows how a more stable condition during such fluctuations improved the overall execution time. *RQ3* was answered by applying different penalty-related constraints in the equations from Sect. 5, demonstrating that the system was able to find a solution which was minimal penalties wise. Conceptually, this positions penalty-based reasoning as an abstraction for balancing operational resilience and cost in self-adaptive DTs, applicable across scenarios.

6.3 Threats to validity

While the reconfiguration of available resources in hospital wards by means of penalty-based reasoning allows the system to adapt to, e.g., crisis scenarios in which the current ward capacity may not suffice for the hospital's needs, it may still not be robust enough for the system to ensure the allocation of patients to bed bays.

The adaptation process improved the responsiveness to a varying number of incoming patients to the ward, even when there were

spikes in seasonal periods. However, the allocation process is not only constrained by the number of patients; other parameters are also taken into account when allocating patients to rooms. As such, the lifecycle management might not be capturing completely edge cases, e.g., too high a number of contagious patients, or patients with special needs that will require rooms that make the solution non-satisfiable. In this regard, considering only the current load of the ward with the percentage load threshold ε might leave out cases in which we allocate enough space to cover the number of patients, but not enough rooms to satisfy all constraints. Moreover, the condition for adaptation might differ in terms of performance (see Sect. 4.2). While our implementation uses periodic adaptation, some applications might require conditional triggers; e.g., in real-time safety-critical systems, where both safety and efficiency is key, the current approach might require further de-coupling to improve the overall performance, as the computational overhead introduced by the semantic lifting might be too high for such systems. However, in terms of flexibility, our system can be extended with different policies for resource management by providing other optimisers.

Solver inaccuracies may also influence the quality of the found bed allocations; machine precision and stochastic solving strategies are known caveats for solving complex optimisation problems. We here used Z3, which implements globally optimal solutions, and did not further investigate the solution quality. Our evaluation assesses architectural feasibility and adaptive behaviour, rather than optimising hospital capacity planning or operational policies.

7 Discussion: Architectural Lessons

This section summarises what we have learnt from the experiments and from integrating semantic reflection, lifecycle management, and penalty-guided optimisation in DynResDT. The lessons concern trade-offs exposed by the architecture behaves under varying load.

Lesson 1: Semantic reflection costs to invest in principled correctness. Semantic reflection and lifecycle management introduce measurable overhead, especially when the twin must realign its internal model frequently. In return, they ensure consistency of the runtime model with the structural representation of the real system and that adaptations are triggered in a timely manner. This is the key architectural trade-off between correctness and runtime efficiency [18].

Lesson 2: Stable demand reduces reflection overhead. When demand is sustained rather than oscillatory, the number of structural updates decreases and the semantic reflection overhead becomes negligible. The bottleneck then shifts to data retrieval and simulation, which can be scaled using standard engineering techniques. This shows that the cost of principled reflection is proportional to how dynamic the environment is, not to the base system size.

Lesson 3: Penalties make resilience–cost trade-off explicit. Penalty-guided optimisation turns the resilience-versus-cost relation into an explicit reasoning element of the architecture. Overflow capacity is opened only when necessary and released once demand falls, allowing the system to balance robustness and efficiency without manual intervention. This mechanism provides a systematic way to capture operational policies as quantitative penalties. In our implementation, fixed penalties reflected the usage status in the ward. However, penalties might also be constructed dynamically, by keeping track of whether a given room is already open, and adjusting the overall penalty to the available bed bays. While this approach can be achieved within our system, it would require the admitting nurse to have knowledge of the optimisation process; this would impose a higher level of complexity, as the admitting nurses might only have knowledge of the domain of the real system.

Implications. Together, these lessons suggest that integrating semantic reflection, lifecycle management, and penalty-guided optimisation provides a principled yet practical way to design self-adaptive digital twins for dynamic resource management. Future work will extend this approach to multi-ward and multi-hospital settings and explore its applicability to other domains.

Architecture applicability. While BedreFlyt is tailored to hospital ward management, the proposed architecture is applicable to other domains where resource layouts must adapt to changing conditions. To this aim, the semantic model, lifecycle conditions, and penalty functions need to capture the target system's characteristics and adaptation goals. The semantic model should capture the essential entities, relationships, and constraints relevant to the domain. Lifecycle conditions should reflect the states and transitions that require adaptation. Penalty functions should quantify the costs associated with different resource configurations and operational states. While the current implementation focuses on simple conditions, the architecture is designed to support more complex adaptation strategies. A key aspect is that the semantic model provides a structured representation of ward state and constraints, which can be extended with richer adaptation logic.

8 Conclusion and Future Work

This paper introduced a self-adaptive DT architecture integrating semantic reflection, lifecycle management, and penalty-guided optimisation for runtime resource reconfiguration. The approach was realised in DynResDT, a prototype for hospital bed allocation.

The experiments demonstrated that the DT maintains runtime alignment with the structural representation of the real system, adapts its structure in response to fluctuating demand, and makes cost-aware allocation decisions. While semantic reflection introduces measurable overhead, this cost is justified by improved correctness and becomes negligible under sustained load.

The study shows how semantic reflection and OMT-based optimisation can be combined in a practical architecture for self-adaptive DTs. Semantic reflection and lifecycle mechanisms provide reliable adaptation triggers, and penalties encode explicit trade-offs between resilience and operational cost that reflect allocation policies.

Our results indicate that combining semantic reflection, lifecycle adaptation, and penalty-guided optimisation is a promising architectural pattern for self-adaptive DTs in the ward setting. Future work will test this in multi-ward federation, explore probabilistic decision models (e.g., MDPs) for branching treatment pathways, automatic selection and runtime learning of penalties and threshold parameters, and assess applicability beyond healthcare.

Acknowledgments

In our implementation, GitHub Copilot was utilised for code completion and rapid prototyping.

References

[1] Ginikachi Anyene, Celeste Schultz, Anthony Nepomuceno, and Inki Kim. 2025. Digital-twin co-simulation framework to support informed decision in healthcare planning and management. *Simul.* 101, 3 (2025), 361–375. doi:10.1177/00375497241283047

[2] Luciano Baresi, Marco Garlini, and Giovanni Quattrocchi. 2025. Dynamic Resource Allocation for Deadline-Constrained Neural Network Training. In *20th IEEE/ACM Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 39–49. doi:10.1109/SEAMS66627.2025.00013

[3] Nelly Bencomo, Sebastian Götz, and Hui Song. 2019. Models@run.time: a guided tour of the state of the art and research challenges. *Softw. Syst. Model.* 18, 5 (Oct. 2019), 3049–3082. doi:10.1007/s10270-018-00712-x

[4] Nikolaj S. Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. *νZ* - An Optimizing SMT Solver. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS15, ETAPS (LNCS, Vol. 9035)*. Springer, 194–199. doi:10.1007/978-3-662-46681-0_14

[5] Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. Models@ run.time. *Computer* 42, 10 (2009), 22–27. doi:10.1109/MC.2009.326

[6] Jane Cleland-Huang, Ankit Agrawal, Michael Vierhauser, Michael Murphy, and Mike Prieto. 2022. Extending MAPE-K to support human-machine teaming. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Pittsburgh, Pennsylvania) *(SEAMS '22)*. Association for Computing Machinery, New York, NY, USA, 120–131. doi:10.1145/3524844.3528054

[7] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS08, ETAPS (LNCS, Vol. 4963)*. Springer, 337–340. doi:10.1007/978-3-540-78800-3_24

[8] Rina Dechter. 2003. *Constraint Processing*. Morgan Kaufmann, San Francisco. 363–397 pages. doi:10.1016/B978-155860890-0/50014-1

[9] Farid Edrisi, Diego Perez-Palacin, Mauro Caporuscio, and Samuele Giussani. 2023. Adaptive Controllers and Digital Twin for Self-Adaptive Robotic Manipulators. In *18th IEEE/ACM Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2023, Melbourne, Australia, May 15-16, 2023*. IEEE, 56–67. doi:10.1109/SEAMS59076.2023.00017

[10] Hao Feng, Cláudio Gomes, Santiago Gil, Peter Høgh Mikkelsen, Daniella Tola, Peter Gorm Larsen, and Michael Sandberg. 2022. Integration Of The Mape-K Loop In Digital Twins. In *Proc. Annual Modeling and Simulation Conf. (ANNSIM 2022)*. IEEE, 102–113. doi:10.23919/ANNSIM55834.2022.9859489

[11] John Fitzgerald, Cláudio Gomes, and Peter Gorm Larsen (Eds.). 2024. *The Engineering of Digital Twins*. Springer. doi:10.1007/978-3-031-66719-0

[12] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. 2010. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press. doi:10.1201/9781420090512

[13] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4, Article 71 (July 2021), 37 pages. doi:10.1145/3447772

[14] Su-Wen Huang, Shao-Jen Weng, Shyue-Yow Chiou, Thi-Duong Nguyen, Chih-Hao Chen, Shih-Chia Liu, and Yao-Te Tsai. 2024. A Study on Decision-Making for Improving Service Efficiency in Hospitals. *Healthcare* 12, 3 (2024), 405. doi:10.3390/healthcare12030405

[15] Eduard Kamburjan, Nelly Bencomo, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. 2024. Declarative Lifecycle Management in Digital Twins. In *Proc. 1st Intl. Conf. on Engineering Digital Twins (EDTconf 2024) (MODELS Companion'24)*. ACM, 353–363. doi:10.1145/3652620.3688248

[16] Eduard Kamburjan, Crystal Chang Din, Rudolf Schlatte, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. 2022. Twinning-by-Construction: Ensuring Correctness for Self-adaptive Digital Twins. In *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13701)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, 188–204. doi:10.1007/978-3-031-19849-6_12

[17] Eduard Kamburjan, Vidar Norstein Klungre, Yuanwei Qu, Rudolf Schlatte, Egor V. Kostylev, Martin Giese, and Einar Broch Johnsen. 2025. Semantically Reflected Programs. *CoRR* abs/2509.03318 (2025). doi:10.48550/ARXIV.2509.03318

[18] Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Einar Broch Johnsen, and Martin Giese. 2021. Programming and Debugging with Semantically Lifted States. In *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12731)*, Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Óscar Corcho, Petar Ristoski, and Mehwish Alam (Eds.). Springer, 126–142. doi:10.1007/978-3-030-77385-4_8

[19] Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Silvia Lizeth Tapia Tarifa, David Cameron, and Einar Broch Johnsen. 2022. Digital Twin Reconfiguration Using Asset Models. In *Proc. 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Practice (ISoLA 2022) (LNCS, Vol. 13704)*. Springer, 71–88. doi:10.1007/978-3-031-19762-8_6

[20] Eduard Kamburjan, Andrea Pferscher, Rudolf Schlatte, Riccardo Sieve, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. 2025. Semantic Reflection and Digital Twins: A Comprehensive Overview. In *The Combined Power of Research, Education, and Dissemination: Essays Dedicated to Tiziana Margaria on the Occasion of Her 60th Birthday*, Mike Hinchey and Bernhard Steffen (Eds.). Lecture Notes in Computer Science, Vol. 15240. Springer, 129–145. doi:10.1007/978-3-031-73887-6_11

[21] Eduard Kamburjan, Riccardo Sieve, Chinmayi Prabhu Baramashetru, Marco Amato, Gianluca Barmina, Eduard Occhipinti, and Einar Broch Johnsen. 2024. GreenhouseDT: An Exemplar for Digital Twins. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2024, Lisbon, Portugal, April 15-16, 2024*, Luciano Baresi, Xiaoxing Ma, and Liliana Pasquale (Eds.). ACM, 175–181. doi:10.1145/3643915.3644108

[22] Åsmund A. A. Kløvstad, Paul Kobialka, Riccardo Sieve, Andrea Pferscher, Laura Slaughter, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen. 2026. What-If Scenarios for the BedreFlyt Digital Twin. In *Principles of formal quantitative analysis — Essays dedicated to Christel Baier on the occasion of her 60th birthday (LNCS)*. Springer Nature Switzerland, 360–381. doi:10.1007/978-3-031-97439-7_18

[23] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. 2018. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine* 51, 11 (2018), 1016–1022. doi:10.1016/j.ifacol.2018.08.474

[24] National Academies of Sciences, Engineering, and Medicine (NASEM). 2024. *Foundational Research Gaps and Future Directions for Digital Twins*. The National Academies Press. doi:10.17226/26894

[25] Marco Picone, Prasad Talasila, Nicola Bicocchi, and Peter Gorm Larsen. 2025. Exploring Devops Practices for Lifecycle Management of Physical and Digital Twins in Cyber-Physical Systems. In *8th IEEE International Conference on Industrial Cyber-Physical Systems, ICPS 2025, Emden, Germany, May 12-15, 2025*. IEEE, 1–6. doi:10.1109/ICPS65515.2025.11087838

[26] Paolo Pileggi, Elena Lazovik, Jeroen Broekhuijsen, Michael Borth, and Jacques Verriet. 2020. Lifecycle Governance for Effective Digital Twins: A Joint Systems Engineering and IT Perspective. In *IEEE International Systems Conference, SysCon 2020, Montreal, QC, Canada, August 24 - September 20, 2020*. IEEE, 1–8. doi:10.1109/SYSCON47679.2020.9275662

[27] Yutao Qi, Dazhuang Liu, Xiaodong Li, Jiaojiao Lei, Xiaoying Xu, and Qiguang Miao. 2020. An adaptive penalty-based boundary intersection method for many-objective optimization problem. *Inf. Sci.* 509 (2020), 356–375. doi:10.1016/J.INS.2019.03.040

[28] Zijie Ren, Jiafu Wan, and Pan Deng. 2022. Machine-Learning-Driven Digital Twin for Lifecycle Management of Complex Equipment. *IEEE Trans. Emerg. Top. Comput.* 10, 1 (2022), 9–22. doi:10.1109/TETC.2022.3143346

[29] Riccardo Sieve, Eduard Kamburjan, Ferruccio Damiani, and Einar Broch Johnsen. 2025. Declarative Dynamic Object Reclassification. In *39th European Conference on Object-Oriented Programming, ECOOP 2025, June 30 to July 2, 2025, Bergen, Norway (LIPIcs, Vol. 333)*, Jonathan Aldrich and Alexandra Silva (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 29:1–29:31. doi:10.4230/LIPICS.ECOOP.2025.29

[30] Riccardo Sieve, Paul Kobialka, Andrea Pferscher, Nelly Bencomo, Silvia Lizeth Tapia Tarifa, Buster Solomon Rasmussen, and Einar Broch Johnsen. 2026. Additional Material for the paper A Self-Adaptive Digital Twin Architecture for Dynamic Resource Management (Version 3). doi:10.5281/zenodo.18340349

[31] Riccardo Sieve, Paul Kobialka, Laura Slaughter, Rudolf Schlatte, Einar Broch Johnsen, and Silvia Lizeth Tapia Tarifa. 2025. BedreFlyt: Improving Patient Flows through Hospital Wards with Digital Twins. In Proceedingsof the First International Workshop on *Autonomous Systems Quality Assurance and Prediction with Digital Twins*, Hamilton, Canada, 4th May 2025 *(Electronic Proceedings in Theoretical Computer Science, Vol. 418)*, Marsha Chechik, Arianna Fedeli, Gianluca Filippone, Federico Formica, Mirgita Frasheri, Nico Hochgeschwender, and Lina Marsso (Eds.). Open Publishing Association, 1–15. doi:10.4204/EPTCS.418.1

[32] Ann-Kathrin Splettstößer, Carsten Ellwein, and Andreas Wortmann. 2023. Self-adaptive digital twin reference architecture to improve process quality. *Procedia CIRP* 119 (2023), 867–872. doi:10.1016/j.procir.2023.03.131 The 33rd CIRP Design Conference.

[33] Biruk G. Tessema and Gary G. Yen. 2006. A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization. In *IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*. IEEE, 246–253. doi:10.1109/CEC.2006.1688315

[34] Adelinde M. Uhrmacher, Peter I. Frazier, Reiner Hähnle, Franziska Klügl, Fabian Lorig, Bertram Ludäscher, Laura Nenzi, Cristina Ruiz Martin, Bernhard Rumpe, Claudia Szabo, Gabriel A. Wainer, and Pia Wilsdorf. 2024. Context, Composition, Automation, and Communication: The C^2AC Roadmap for Modeling and Simulation. *ACM Trans. Model. Comput. Simul.* 34, 4 (2024), 23:1–23:51. doi:10.1145/3673226

[35] Thomas Vogel. 2018. mRUBiS: an exemplar for model-based architectural self-healing and self-optimization. In *Proceedings of the 13th International Conference*

on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2018, Gothenburg, Sweden, May 28-29, 2018, Jesper Andersson and Danny Weyns (Eds.). ACM, 101–107. doi:10.1145/3194133.3194161

[36] Yuping Wang and Wei Ma. 2006. A Penalty-Based Evolutionary Algorithm for Constrained Optimization. In *Advances in Natural Computation, Second International Conference, ICNC 2006, Xi'an, China, September 24-28, 2006. Proceedings, Part I (Lecture Notes in Computer Science, Vol. 4221)*, Licheng Jiao, Lipo Wang, Xinbo Gao, Jing Liu, and Feng Wu (Eds.). Springer, 740–748. doi:10.1007/11881070_99

[37] Gereon Weiss, Philipp Schleiss, Daniel Schneider, and Mario Trapp. 2018. Towards integrating undependable self-adaptive systems in safety-critical environments. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2018, Gothenburg, Sweden, May 28-29, 2018*, Jesper Andersson and Danny Weyns (Eds.). ACM, 26–32. doi:10.1145/3194133.3194157

[38] Nimnath Withanachchi, Yasuo Uchida, Shyama Nanayakkara, Dulani Samaranayake, and Akiko Okitsu. 2007. Resource allocation in public hospitals: Is it effective? *Health Policy* 80, 2 (2007), 308–313. doi:10.1016/j.healthpol.2006.03.014

[39] Md Doulotuzzaman Xames and Taylan G. Topcu. 2023. Toward Digital Twins for Human-in-the-loop Systems: A Framework for Workload Management and Burnout Prevention in Healthcare Systems. In *IEEE 3rd International Conference on Digital Twins and Parallel Intelligence, DTPI 2023, Orlando, FL, USA, November 7-9, 2023*. IEEE, 1–6. doi:10.1109/DTPI59677.2023.10365449