Approche DevOps pour le développement et l'évolution systématique des jumeaux numériques appliqués aux environnements bâtis

par

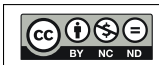
Sara AISSAT

MÉMOIRE PAR ARTICLES PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE AVEC MÉMOIRE EN GÉNIE LOGICIEL
M. Sc. A.

MONTRÉAL, LE 28 AVRIL 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

Sara Aissat, 2025



Cette licence Creative Commons signifie qu'il est permis de diffuser, d'imprimer ou de sauvegarder sur un autre support une partie ou la totalité de cette oeuvre à condition de mentionner l'auteur, que ces utilisations soient faites à des fins non commerciales et que le contenu de l'oeuvre n'ait pas été modifié.

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Francis Bordeleau, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Érik Poirier, codirecteur
Département de génie de la construction à l'École de technologie supérieure

M. Julien Gascon-Samson, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Fabio Petrillo, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 22 AVRIL 2025

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AVANT-PROPOS

Au cours de la rédaction de ce mémoire, deux publications ont été réalisées et présentées lors de conférences, lesquelles sont listées en annexe.

CIB W78 and buildingSMART International Summit : L'article intitulé *« A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins »* (Aissat, Beaulieu, Bordeleau, Motamedi & Poirier) a été présenté lors de la 41^{e} conférence internationale CIB W78, organisée du 1er au 3 octobre 2024 à Marrakech, Maroc.

International Conference on Engineering DTs (EDTconf) : L'article *« JuNo-OPS : A DevOps Framework for the Engineering of Digital Twins for Built Assets »* (Aissat *et al.*, 2024a) a été présenté lors de la conférence EDT, tenue à Linz, en Autriche les 23 et 24 septembre 2024. À la suite de cette présentation, une invitation a été reçue pour soumettre une version étendue à la revue scientifique *Software and Systems Modeling* (SoSyM), actuellement en cours d'évaluation. Cette version étendue, qui fait l'objet de ce mémoire, est en cours de modification en réponse aux commentaires reçus durant le processus de revue.

Ces travaux illustrent l'évolution et la diffusion des résultats de ce mémoire au sein de la communauté scientifique internationale.

REMERCIEMENTS

Je tiens avant tout à exprimer ma profonde gratitude à mon directeur, Francis Bordeleau, qui m'a d'abord initié au DevOps et ensuite à sa passion pour les jumeaux numériques. Grâce à vous, un chemin s'est tracé devant moi, dans lequel nous avons mis en place une vision claire, et qu'il reste encore à faire évoluer et enrichir sur de belles années à venir. Je remercie également mon co-directeur, Erik Poirier, avec qui nous avons su mêler nos expertises pour nourrir mes apprentissages. Vous avez été de précieux conseils, toujours présent pour apporter votre vision du monde de la construction, une vision qui a été essentielle à ma démarche. À l'apport aussi d'Ali Motamedi, au cœur des travaux menés sur le GRIDD. Votre implication et vos initiatives ont constitué une base essentielle de mon travail et ont grandement contribué à nourrir mes recherches.

Je n'ai cessé de me répéter ma chance et ma reconnaissance d'avoir pu travailler aux côtés de superviseurs et collègues aussi aidants, compréhensifs et motivants au quotidien. Je pense qu'avec une bonne équipe on peut accomplir de grandes choses, et nous avons là une bonne équipe, alors les grands accomplissements sont à venir. Je vous remercie pour la confiance que vous m'avez accordée, de m'avoir encouragée et poussée vers des exploits qui m'étaient inimaginables ; aller présenter mon travail à l'étranger, dans ma troisième langue !

Je tiens également à remercier mes proches, ma famille et mes amis, qui m'ont supportés à travers mes études, et bien sûr, enduré mes innombrables angoisses. À ma petite maman, qui a toujours été là pour écouter mes tracas, dont la patience m'a fait un bien immense. À ma sœur de cœur, Samia et ma sœur Rima, qui m'aviez écoutée parler d'informatique, sans tout comprendre, mais qui avez su partager le poids de mon travail avec moi. À vous tous, je dédie une part de ce succès ; une part seulement, car il faut tout de même admettre que c'est moi qui ai rédigé ce mémoire :).

Un grand merci aussi à mon père, pour m'avoir transmis cette voie de l'informatique et pour la sagesse de la vie et du travail inculquée.

Enfin, un merci spécial à mon précieux ami Rakeym, sans qui l'ETS n'aurait pas connu mon passage. Je te serai à jamais entièrement reconnaissante pour ces années d'équipe redoutable que nous avons formé tout au long de notre scolarité. Je te remercie aussi de m'avoir fait choisir l'ETS et l'ingénierie avec toi ; ça ne m'a pas déçue, bien qu'on aurait dû finir ce parcours à deux. J'ai le regret que nous ne connaîtrons jamais « Ing. Rakeym Jean Baptiste », mais compte sur moi je travaillerais pour deux ; toi, repose en paix.

Je conclus ces remerciements en espérant que ce travail soit à la hauteur des attentes emises à mon égard.

Approche DevOps pour le développement et l'évolution systématique des jumeaux numériques appliqués aux environnements bâtis

Sara AISSAT

RÉSUMÉ

Au cours de la dernière décennie, les jumeaux numériques ont connu un essor remarquable, porté par leur fort potentiel et leur impact tangible dans divers secteurs, notamment celui de l'environnement bâti, où ils offrent des perspectives inédites en termes d'optimisation de la gestion des actifs et d'amélioration de l'efficacité opérationnelle. Leur mise en œuvre repose, toutefois, sur des systèmes logiciels complexes, nécessitant non seulement une intégration rigoureuse, mais aussi une maintenance continue et une capacité d'évolution soutenue pour s'adapter aux exigences changeantes des utilisateurs.

Le développement des jumeaux numériques, malgré les avancées dans le domaine, reste freiné par l'absenced'approches systématiques et évolutives. Ce manque se révèle d'autant plus critique que leur mise en œuvre exige une collaboration interdisciplinaire, mobilisant des expertises variées. En ingénierie logicielle, le paradigme DevOps s'est quant à lui imposé comme un levier stratégique pour améliorer la qualité et la fiabilité des systèmes, grâce à l'adoption de bonnes pratiques issues du génie logiciel appliquées à l'ensemble du cycle de vie des applications.

Le présent mémoire propose une approche DevOps adaptée aux jumeaux numériques dans le contexte de l'environnement bâti. Elle vise à encourager un développement agile et itératif, capable de générer rapidement de la valeur tout en assurant une évolution continue des systèmes, notamment grâce à l'automatisation des processus à travers des pipelines d'intégration et de déploiement continus (CI/CD). JuNo-OPS, un *framework* conçu selon une architecture microservices, favorisant la modularité, la scalabilité et l'indépendance des services, est ainsi présenté. Il repose sur une infrastructure pensée pour abstraire les aspects techniques transversaux du développement, permettant aux experts du domaine de se concentrer sur l'apport fonctionnel des jumeaux numériques spécifiques à leurs cas d'usage, qu'il s'agisse de gestion énergétique, de confort thermique, de qualité de l'air ou tout autre besoins métiers.

L'approche proposée s'appuie sur l'observation et l'analyse des développements de divers jumeaux numériques dans le domaine de l'environnement bâti, mettant en lumière les pratiques actuelles ainsi que les principaux défis associés. Le *framework* a été développé, testé et exploré dans le contexte d'une salle multifonctionnelle à l'ÉTS. Il s'articule autour de deux aspects principaux : l'architecture modulaire du logiciel et l'infrastructure DevOps mise en place pour soutenir son développement, son intégration continue et son déploiement continu. Enfin, ce mémoire aborde les défis rencontrés ainsi que les perspectives d'évolution du *framework*, visant à en améliorer l'efficacité et à élargir son champ d'application.

Mots-clés: Jumeau numérique (JN), DevOps, actifs/environnements bâtis, BIM, IoT, microservices, méthodologie logicielle, pipelines CI/CD

A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins

Sara AISSAT

ABSTRACT

Over the past decade, digital twins have experienced remarkable growth, driven by their high potential and tangible impact across various sectors, particularly in the built environment, where they open new possibilities for optimizing asset management and enhancing operational efficiency. However, deploying digital twins relies on complex software systems that require not only rigorous integration but also ongoing maintenance and the ability to evolve continuously to meet changing user requirements.

Despite progress in this field, the development of digital twins remains hindered by the lack of systematic and scalable approaches. This gap is especially critical given the interdisciplinary collaboration required, drawing on a wide range of expertise. In software engineering, the DevOps paradigm has emerged as a key strategy to improve system quality and reliability, by applying best practices from software development across the entire application lifecycle.

This thesis introduces a DevOps-based approach tailored for digital twins in the built environment. The goal is to foster agile, iterative development that can quickly deliver value while supporting the continuous evolution of systems, particularly through the automation of integration and deployment processes (CI/CD pipelines). It presents JuNo-OPS, a framework built on a microservices architecture designed to promote modularity, scalability, and service independence. The framework relies on an infrastructure that abstracts away cross-cutting technical concerns, enabling domain experts to focus on the functional contributions of digital twins tailored to their specific use cases, whether in energy management, thermal comfort, air quality, or other operational needs.

The proposed approach is grounded in the observation and analysis of various digital twin developments within the built environment, highlighting current practices and key challenges. The framework was developed, tested, and explored in the context of a multifunctional room at ÉTS. It is structured around two main pillars : the modular software architecture and the DevOps infrastructure supporting continuous development, integration, and deployment. Finally, the thesis discusses the challenges encountered and outlines future directions for enhancing the framework's effectiveness and expanding its applicability.

Keywords: DevOps, digital twin (DT), built assets, BIM, IoT, microservices, software methodology, CI/CD pipelines

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ACC	Autodesk Construction Cloud
API	Interfaces de programmation d'applications
AT	Actual Twin
BIM	Building Information Modeling
BLE	Bluetooth Low Energy
CD	Continuous Deployment
CDT	Cognitive Digital Twin
CI	Continuous Integration
CIM	City Information Modeling
CPS	Systèmes cyber-physiques
DORA	DevOps Research and Assessment
DSR	Design Science Research
DT	Digital Twin
DTaaS	Digital Twin as a Service
EMS	Energy Management System
ETL	Extract, Transform, Load
ÉTS	École de Technologie Supérieure
FDD	Fault Detection and Diagnosis
F&CF	Fast and Continuous Feedback
GRIDD	Groupe de Recherche en Intégration et Développement Durable
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
IA	Intelligence artificielle
IoT	Internet des objets
IVHM	Integrated Vehicle Health Management

JN	Jumeaux numériques
LP-WAN	Low Power Wide Area Network
MaC	Monitoring as Code
MBSE	Model-Based Systems Engineering
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MPC	Model Predictive Control
NASA	National Aeronautics and Space Administration
O&M	Operation and Maintenance
PFE	Projet de fin d'études
PLM	Product Lifecycle Management
PMV	Predicted Mean Vote
PoC	Proof of Concept
PPD	Predicted Percentage of Dissatisfied
QoS	Quality of Service
RAMI4.0	Reference Architecture Model Industrie 4.0
RPi	Raspberry Pi
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SOA	Service-Oriented Architecture
SoS	System of Systems
SSN	Semantic Sensor Network

INTRODUCTION

Au cœur des grandes avancées technologique de notre ère, les données sont une composante essentielle du pilotage des systèmes modernes, qu'il s'agisse de villes intelligentes (Lom, Pribyl & Svitek, 2016), de véhicules autonomes (Yaqoob *et al.*, 2019), d'industries manufacturières (Tien, 2017), de l'aérospatiale (Si & Baier, 2015) ou encore de la prise de décision stratégique et de la gestion algorithmique (Ikemoto & Marsh, 2007; Trabucchi, Buganza & Pellizzoni, 2017; Chen & Hsieh, 2014). La collecte de données pouvant être analysées, traitées et exploitées dans le but d'être transformées en modèles mathématiques, computationnels et algorithmiques est essentielle à ces domaines en pleine évolution (Moeller, 2003), facilitant ainsi la compréhension et la prédiction des comportements de leurs reseaux d'intérêts (Lom *et al.*, 2016). Cette transformation des données en modèles divers ouvre la voie à des représentations numériques dynamiques et personnalisées, adaptées aux besoins spécifiques des utilisateurs et des systèmes qu'elles reflètent (Tien, 2017).

Les jumeaux numériques (JN) incarnent cette évolution en permettant la modélisation dynamique de systèmes grâce à l'intégration de données en temps réel (Tao, Zhang, Liu & Nee, 2018). Ils se sont imposés, depuis quelques années, comme une approche incontournable dans divers champs d'application, offrant des opportunités significatives, mais soulevant également de nombreux défis techniques (Zheng, Lu & Kiritsis, 2022). Un jumeau numérique est bien plus qu'un simple outil de collecte de données. Il s'agit d'un système logiciel sophistiqué qui transforme ces données en informations actionnables. Il facilite la prise de décision, améliore l'efficacité des processus (Jones, Snider, Nassehi, Yon & Hicks, 2020) et permet d'accomplir des tâches plus rapidement, avec plus de précision, ou même de réaliser ce qui ne serait pas possible sans une telle technologie (Lu *et al.*, 2020).

Dans l'environnement bâti, ces systèmes virtuels, capables de refléter et d'anticiper le comportement de leurs homologues physiques, offrent des opportunités immenses pour optimiser

la gestion des actifs tout au long de leur cycle de vie, en permettant le suivi et l'amélioration de divers aspects des infrastructures à chaque étape de leur évolution (Jones *et al.*, 2020). Les JN d'actifs bâtis, traitent des aspects tel que l'amélioration de l'efficacité énergétique (Merabet *et al.*, 2021), ou encore le confort des occupants passant par la qualité de l'aire (Chenari, Carrilho & Da Silva, 2016), permettant de réaliser des économies substantielles en termes de coûts d'exploitation (Khajavi, Motlagh, Jaribion, Werner & Holmström, 2019). Les jumeaux numériques regroupent plusieurs disciplines et technologies, notamment l'Internet des objets (IoT), qui joue un rôle clé dans la collecte de données (Davari, Shahinmoghadam, Motamedi & Poirier, 2022). Lorsqu'il s'agit d'actifs bâtis, d'autres systèmes connexes viennent s'y ajouter, comme le Building Information Modeling (BIM), qui doivent être intégrées de manière cohérente (Davari *et al.*, 2022). Cette complexité intrinsèque liée à l'accumulation et à la gestion de vastes quantités de données, rend leur conception particulièrement exigeante.

Définir précisément ce qu'est un jumeau numérique constitue déjà un défi en soi. Au-delà de cette définition, tracer un cadre structurant pour les jumeaux numériques appliqués aux environnements bâtis est impératif afin de simplifier leur mise en place et de faciliter leur évolution. Actuellement, la majorité des développements de jumeaux numériques dans le secteur du bâti se font dans des contextes de recherche, sous forme de prototypes et de preuves de concept (*Proof of Concept*, PoC) (Singh *et al.*, 2021), explorant continuellement de nouvelles approches pour en saisir le bénéfice d'une part, et pour faciliter leur mise en œuvre d'une autre part. Si ces initiatives permettent de démontrer la valeur ajoutée des JN, elles restent souvent *ad hoc* [1], ne suivant pas de méthodologie rigoureuse ni d'architecture bien définie. Cela entraîne, entre autres, des problèmes de mise à l'échelle et de reproductibilité des approches de développement, mais surtout un manque de cohérence et de prévisibilité dans la planification des ressources et la gestion des résultats (Shahzad, Shafiq, Douglas & Kassem, 2022). Finalement, plusieurs

[1] Le terme ad hoc désigne une approche ponctuelle ou spécifique, développée pour répondre à un besoin immédiat sans cadre méthodologique formel ni standardisation.

outils ont émergé pour répondre à certains défis spécifiques, mais aucun ne couvre l'ensemble du processus de développement de manière intégrée, ce qui permettrait également d'éviter des obstacles non anticipés.

Face à ces constats, cette recherche propose d'appliquer une approche DevOps au développement des jumeaux numériques. Un ensemble de méthodes et d'outils issus du génie logiciel visant à optimiser les processus de développement, d'intégration et de déploiement logiciels en favorisant l'automatisation, la gestion efficace du cycle de vie des applications et une meilleure orchestration des flux de travail (Kim, Humble, Debois, Willis & Forsgren, 2021). Une tâche qui s'avère laborieuse en raison du regroupement de systèmes, de technologies, d'outils et d'expertises issues à la fois du génie logiciel et de celui de la construction. Cette approche permettrait, dans le contexte des JN, de gérer la complexité multidisciplinaire en offrant une infrastructure facilitant la collaboration entre les experts métiers (par exemple ceux spécialisés en confort thermique, qualité de l'air, gestion énergétique, etc.) leur permettant de contribuer au développement des services offerts par le jumeaux numérique sans qu'ils aient à se soucier des contraintes techniques sous-jacentes.

Afin de poser les bases concrètes de cette approche, trois contributions principales ont été réalisées dans le cadre de cette recherche :

- **La mise en place d'un *framework* DevOps** : JuNo-OPS, un *framework* DevOps conçu pour structurer le développement des jumeaux numériques en environnement bâti, est donc né dans cette perspective de concrétiser cette approche. Élaboré à travers le développement d'un jumeau numérique d'une salle multifonctionnelle, il repose sur une infrastructure de développement organisée dans GitLab, incluant la structuration en groupes, la définition des modèles de branches, et la base des dépôts. Cette infrastructure est elle-même conçue selon une architecture bien définie, favorisant la séparation des préoccupations (*Separation of Concerns*). Concrètement, les ingénieurs logiciels sont responsables de la mise en place

d'une infrastructure robuste et scalable, tandis que les experts métier peuvent se concentrer sur le développement de nouvelles fonctionnalités directement en lien avec leur champ d'expertise. JuNo-OPS, appliqué à une infrastructure GitLab de développement s'appuie notamment sur des pipelines d'intégration et de déploiement continus (CI/CD), garantissant une mise en production rapide et efficace des nouveaux développements.

- **La conception d'une architecture modulaire de référence** : L'architecture adoptée repose sur un modèle en microservices, garantissant une évolutivité accrue et facilitant, entre autres, l'intégration des données issues de divers capteurs, tout en minimisant le couplage entre les composants. Elle offre également un accès aux données doté d'une abstraction de la gestion logicielle, simplifiant ainsi le développement des services. Ces services, modélisés en termes d'aspects de jumeaux numériques, comme le confort thermique, sont divisés en petits composants, permettant des avancées rapides et assurant une bonne scalabilité. Les contraintes spécifiques rencontrées par les experts métiers non techniques sont également prises en compte par le DevOps, qui, par sa méthodologie logicielle, intègre les principes agiles et de développement itératif (Kim *et al.*, 2021). Cela permet un suivi continu des avancées tout en autorisant des ajustements rétroactifs pour corriger d'éventuelles erreurs, évitant ainsi l'accumulation d'une quelconque dette technique [2].
- **La validation du Framework JuNo-OPS** : JuNo-OPS a été développé et testé dans le contexte du jumeau numérique du laboratoire du GRIDD (Groupe de Recherche en Intégration et Développement Durable en environnement bâti) à l'École de technologie supérieure (ÉTS), un espace multifonctionnel équipé de capteurs et de systèmes de gestion avancés. Ce travail s'appuie sur des projets antérieurs réalisés au GRIDD, qui ont permis de mettre en place un premier jumeau numérique. Ce dernier a servi de base pour affiner et valider l'approche et

[2] La dette technique désigne l'ensemble des compromis faits lors du développement logiciel qui, bien que permettant des avancées rapides à court terme, peuvent entraîner des coûts élevés en maintenance et corrections ultérieures s'ils ne sont pas résolus progressivement (Ozkaya, Kruchten, Nord & Brown, 2011).

son *framework*, qui désormais est utilisé par des équipes travaillant sur de nouveaux aspects du jumeau numérique, analysant ses effets et identifiant des améliorations continues.

Les travaux présentés dans ce mémoire s'inscrivent dans une continuité de recherches ayant donné lieu à plusieurs publications, dont les références complètes sont fournies en annexe. Une première contribution, intitulée « *A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins* » (Aissat *et al.*), a été présentée lors de la 41e conférence internationale CIB W78 [3], introduisant l'application d'une approche DevOps au développement des jumeaux numériques dans l'environnement bâti. Cette première publication a mis en lumière l'importance d'une ingénierie logicielle systématique et évolutive pour ces systèmes. Cette réflexion a été approfondie à travers une seconde publication, « *JuNo-OPS : A DevOps Framework for the Engineering of Digital Twins for Built Assets* » (Aissat *et al.*, 2024a), présentée à la conférence EDT [4], où a été introduit JuNo-OPS. Cette contribution a permis de formaliser une infrastructure DevOps basée sur une architecture microservices adaptée favorisant une évolution systématique des JN tout en facilitant la collaboration entre experts de différents domaines. Enfin, cette démarche a conduit à une publication en cours de révision dans le journal *Software and Systems Modeling* (SoSyM), proposant une version étendue et approfondie du *framework*. Cette dernière contribution présente l'évolution de JuNo-OPS, intégrant des décisions architecturales affinées, une mise en œuvre consolidée ainsi qu'une validation expérimentale à travers plusieurs projets de jumeaux numériques à l'ÉTS.

Le présent document se structure donc comme suit : une revue de la littérature (chapitre 1) permet d'introduire et de mettre en perspective les différents concepts fondamentaux mobilisés dans cette recherche, notamment ceux liés aux jumeaux numériques et aux pratiques DevOps, tout en mettant en lumière les enjeux spécifiques à leur application dans le secteur de la construction.

[3] CIB W78 and buildingSMART International Summit, 1er au 3 octobre 2024, Marrakech, Maroc.

[4] International Conference on Engineering Digital Twins (EDTconf), 23-24 septembre 2024, Linz, Autriche.

La méthodologie (chapitre 2) présente ensuite les travaux réalisés au sein du GRIDD, terrain d'expérimentation de cette recherche, ainsi que les observations et entretiens menés auprès des acteurs impliqués dans le développement des jumeaux numériques afin de mieux comprendre les pratiques actuelles et les défis rencontrés. Enfin, le cœur de ce travail expose la solution proposée (chapitre 3), suivie d'une discussion des résultats obtenus et des perspectives pour les travaux futurs.

CHAPITRE 1

REVUE DE LITTÉRATURE

Les jumeaux numériques, une technologie émergente apparue au début des années 2000, ont depuis lors révolutionné divers secteurs industriels grâce à leur capacité à représenter des systèmes physiques dans un espace virtuel. Le concept a évolué au fil du temps, incorporant des avancées technologiques majeures, telles que l'IoT, l'intelligence artificielle (IA) et la simulation multiphysique, pour offrir une représentation dynamique et en temps réel de systèmes complexes. En ce qui concerne l'environnement bâti, les jumeaux numériques jouent un rôle clé dans la gestion des infrastructures, notamment à travers la maintenance prédictive et l'optimisation énergétique. Cependant, malgré leur potentiel, l'intégration des jumeaux numériques dans des systèmes complexes demeure un défi, en raison des lacunes dans l'interopérabilité, de la gestion des données et des infrastructures. Ainsi, les efforts pour formaliser une approche commune et développer des outils adaptés à cette technologie se multiplient, tout en explorant l'intégration de pratiques telles que le DevOps pour assurer une gestion fluide et continue des jumeaux numériques.

1.1 Jumeaux numériques : définition

Les jumeaux numériques se sont progressivement imposés comme une technologie clé dans divers secteurs industriels. Leur première apparition remonte à 2003, lorsque Michael Grieves, lors d'une conférence sur la gestion du cycle de vie des produits (PLM), proposa le concept de jumeau numérique comme étant une représentation virtuelle d'un produit physique, liée à ce dernier par un échange continu de données (Grieves, 2023). Cette définition, bien que fondatrice, manquait toutefois de précision et d'uniformité, notamment en ce qui concerne les modalités d'intégration entre les dimensions physique et virtuelle. En effet, elle se limitait à décrire un modèle numérique permettant de suivre l'état d'un objet physique en temps réel, s'appuyant sur des données collectées à partir de capteurs et d'autres dispositifs de suivi.

Évolution du concept

Ce n'est qu'au fil du temps, notamment grâce aux progrès technologiques dans des domaines tels que l'IoT, les données massives et la simulation multi-physiques, que les jumeaux numériques ont connu un essor considérable. En 2012, la National Aeronautics and Space Administration (NASA) a affiné la définition de ce concept, le présentant comme une simulation multiphysique, multiéchelle et probabiliste d'un système, intégrant les meilleurs modèles physiques disponibles en plus des données historiques et en temps réel afin de refléter l'état du dit « jumeau volant » d'un objet physique avec une grande précision (Glaessgen & Stargel, 2012). Ayant pour contexte les véhicules, ce modèle ultra-réaliste est présenté comme pouvant inclure plusieurs systèmes interconnectés du véhicule, tels que la structure, la propulsion, le stockage d'énergie et les systèmes de gestion de la santé du véhicule (*Integrated Vehicle Health Management,* IVHM), permettant ainsi une prévision continue de l'état du véhicule, de sa durée de vie restante et de la probabilité de succès de la mission (Glaessgen & Stargel, 2012). Ce modèle enrichi permet ainsi de simuler un objet physique à différents niveaux de détail, selon les exigences spécifiques de divers domaines d'application.

L'évolution continue du concept a, malgré cela, donné naissance à plusieurs définitions et approches, chacune visant à répondre aux besoins particuliers de secteurs industriels variés. En particulier, dans le domaine de l'industrie manufacturière, les jumeaux numériques se sont rapidement imposés comme des outils permettant de simuler des produits et des processus, d'optimiser la production et de gérer la maintenance prédictive. Cette adoption généralisée a conduit à une diversité de définitions et de perspectives sur les jumeaux numériques, selon les domaines d'application et les besoins spécifiques. Ainsi, plusieurs chercheurs ont présenté différentes façons de concevoir et de mettre en œuvre les jumeaux numériques, en distinguant diverses définitions de base du concept, afin d'en élaborer des formulations plus complexes et englobant plusieurs dimensions.

1.1.1 Les niveaux d'intégration des jumeaux numériques

Les auteurs Kritzinger et al. (2018) définissent les jumeaux numériques selon trois niveaux d'intégration, permettant de classifier les concepts en fonction de la manière dont les données circulent entre le monde physique et le monde numérique, tel qu'illustré à la Figure 1.1 (Kritzinger, Karner, Traar, Henjes & Sihn, 2018). Cette classification permet d'évaluer la maturité des solutions développées et de mieux comprendre les limites des différentes approches.

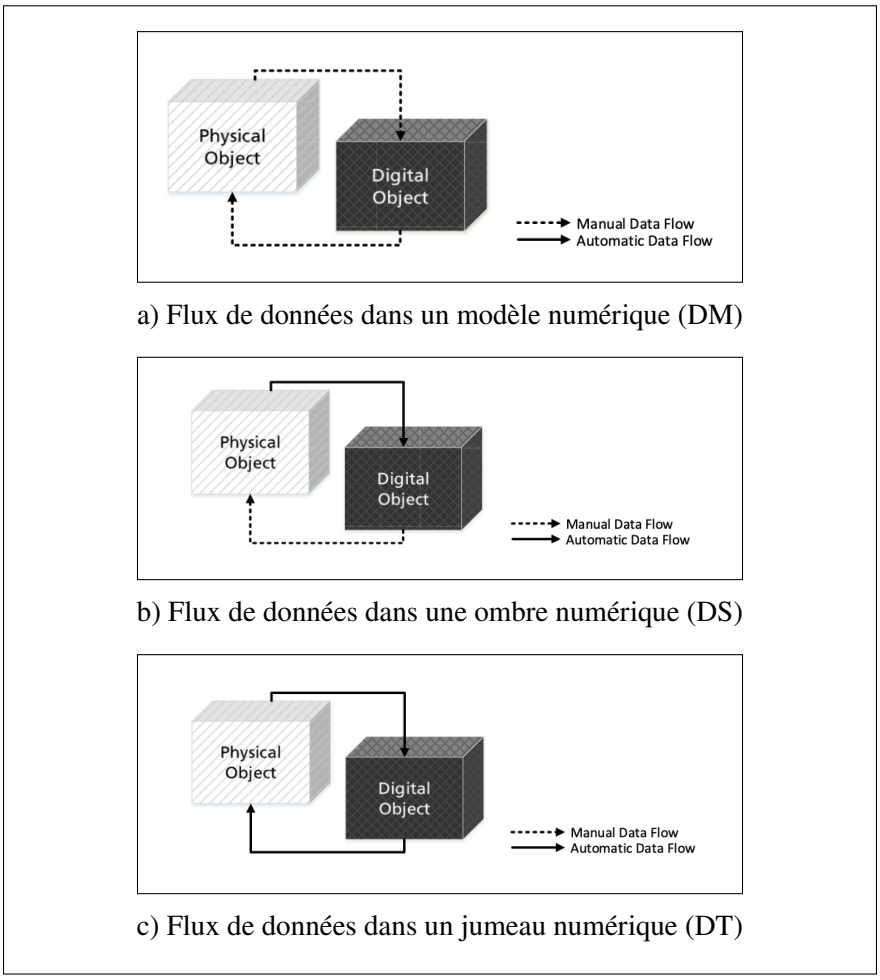


Figure 1.1 Les trois niveaux d'intégration des jumeaux numériques en fonction de la circulation des données entre le monde physique et le monde numérique
Tirée de Kritzinger et al. (2018)

1. **Le modèle numérique (*Digital Model*, DM)** : Ce niveau représente une entité numérique d'un objet physique existant ou planifié, ayant toutefois aucune forme d'échange automatique de données. Les changements dans l'état de l'objet physique n'ont aucun impact direct sur le modèle numérique, et inversement.
2. **L'ombre numérique (*Digital Shadow*, DS)** : À ce niveau, un flux de données unidirectionnel automatique existe entre l'objet physique et son objet numérique. Ainsi, un changement dans l'état de l'objet physique se reflète dans l'ombre numérique, mais l'inverse n'est pas vrai. Ce modèle permet de suivre l'état du système en temps réel sans interaction directe avec le monde physique.
3. **Le jumeau numérique (*Digital Twin*, DT)** : Ce niveau représente l'intégration complète des données entre l'objet physique et son jumeau numérique, avec un échange bidirectionnel en temps réel. L'objet numérique peut également agir comme une instance de contrôle pour l'objet physique, influençant ainsi son état et permettant une orchestration optimale de l'ensemble du système de production. Ce niveau est le plus avancé, où l'objet numérique devient une réplique fonctionnelle de l'objet physique, intégrée à son environnement et capable d'effectuer des actions basées sur l'analyse des données.

Cette classification permet de mieux comprendre les différents niveaux de sophistication des jumeaux numériques, en fonction du degré d'automatisation et d'intégration des données entre les objets physiques et numériques. Kritzinger et al. (2018) insistent sur le fait que les recherches sur le stade le plus avancé, le jumeau numérique complet, sont encore relativement rares, tandis que celles sur les modèles numériques et les ombres numériques sont plus fréquentes dans la littérature (Kritzinger *et al.*, 2018).

1.1.2 Les axes des jumeaux numériques

En complément de cette classification, Tao et al. (2018) proposent une structuration des jumeaux numériques selon cinq dimensions essentielles : l'entité physique (*physical entity* - PE), l'entité virtuelle (*virtual entity* - VE), les connexions (*connection* - CN), les données (*data* - DD) et

les services (services pour le PE et le VE - Ss) (Tao *et al.*, 2018). Ces dimensions, également importantes, définissent la manière dont un jumeau numérique est conçu et exploité, en intégrant les interactions entre les différentes composantes du système, comme illustré à la Figure 1.2.

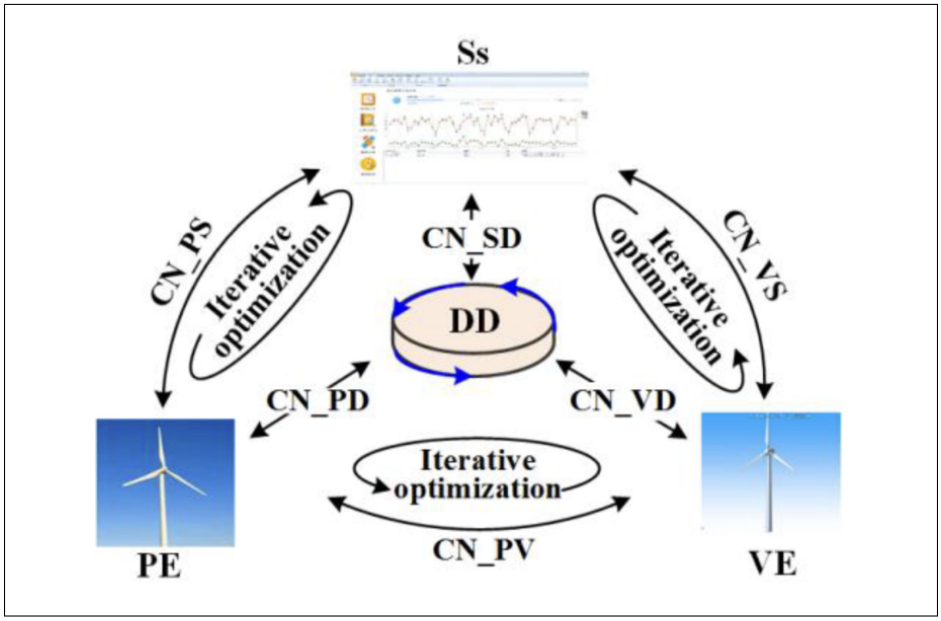


Figure 1.2 Le modèle en cinq dimensions des jumeaux numériques
Tirée de Tao et al. (2018)

Le développement de ces jumeaux numériques représentent une technologie prometteuse pour l'industrie, intégrant diverses disciplines telles que les sciences de l'information, la science des données et l'informatique. Cette approche se structure, selon Tao et al. (2018), autour de quatre grands axes :

1. **la modélisation du jumeau numérique, la simulation, et la vérification, validation et accréditation (VV&A)** : Il est question ici de créer une réplique numérique d'un objet physique, de simuler son comportement dans diverses conditions et de garantir la précision et la fiabilité du modèle numérique par des processus de validation.
2. **La fusion des données** : Englobant trois processus distincts (*data preprocessing*, *data mining*, *data optimization*), cet axe consiste à intégrer des données provenant de différentes

sources, permettant ainsi d'améliorer la précision du modèle numérique et d'assurer que les décisions prises à partir de ce modèle reposent sur des informations complètes et actualisées.

3. **L'interaction et la collaboration** : La résolution de problèmes complexes dans le cadre des jumeaux numériques nécessitant la communication entre ses différentes entités, trois types d'interactions sont sollicitées (*physical-physical*, *virtual-virtual*, et *physical-virtual*). Ensemble, ces interactions permettent une réaction rapide aux changements inattendus, favorisée par un échange constant d'informations entre les espaces physique et virtuel.
4. **Les services associés** : Cet axe englobe l'application des jumeaux numériques dans des services pratiques, tels que la maintenance prédictive et l'optimisation des processus industriels, visant à améliorer les performances et l'efficacité des systèmes. Ces services reposent sur plusieurs théories, telles que l'encapsulation des services, la recherche et la correspondance des services, la modélisation et l'évaluation de la qualité des services (*Quality of Service*, QoS), l'optimisation des services et la gestion de la tolérance aux pannes.

Ces éléments permettent d'optimiser l'intégration des jumeaux numériques dans des systèmes complexes, facilitant ainsi la prise de décision, l'amélioration des performances et l'efficacité des processus industriels (Tao *et al.*, 2018).

1.1.3 Une taxonomie émergente

La diversité des approches et des applications des jumeaux numériques a conduit à des divergences dans leur définition et leur utilisation. Les domaines d'application étant multiples, allant de la fabrication à l'aérospatiale, en passant par la santé et l'agriculture, chaque secteur a proposé une interprétation spécifique du concept. Cette diversité a engendré des variations dans les termes et les définitions utilisées, ce qui a rendu difficile la compréhension du concept dans son ensemble. Afin de répondre à cette problématique, plusieurs chercheurs ont cherché à établir une base commune, en consolidant les différentes perspectives et en identifiant des éléments récurrents à travers la littérature.

Ainsi, pour rassembler ces différentes perspectives, l'article de Jones et al. (2020) a identifié 19 thèmes récurrents dans la littérature, servant à caractériser et définir les jumeaux numériques (Jones *et al.*, 2020). Ces thèmes couvrent des aspects fondamentaux tels que l'entité physique et l'entité virtuelle, les environnements physiques et virtuels, ainsi que les processus associés, les connexions entre ces entités, les états, et bien d'autres. Ces éléments forment ainsi un cadre de plus en plus détaillé et nuancé du jumeau numérique, permettant de mieux comprendre sa complexité et son application dans divers secteurs (Jones *et al.*, 2020).

En parallèle, d'autres taxonomies, telles que celle présentée par Van der Valk et al. (2020), ont également permis d'éclairer la diversité des définitions des jumeaux numériques en les classant selon plusieurs dimensions (Van der Valk *et al.*, 2020). Ces dimensions, détaillées par diverses caractéristiques, ont émergé de multiples définitions officielles, notamment celle de Tao et al. (2018) mentionnée précédemment, ainsi que de plus d'une centaine de publications passées en revue (Tao *et al.*, 2018). Les caractéristiques, identifiant des éléments essentiels tels que l'intégration bidirectionnelle, sont ensuite mises en correspondance avec ces définitions afin d'en définir leur exclusivité. Ces classifications, bien qu'elles ne résolvent pas toutes les divergences, offrent un cadre d'analyse utile pour naviguer à travers les multiples définitions et usages du concept. Ces efforts pour formaliser le concept contribuent à la création d'une base commune, facilitant ainsi les échanges intersectoriels et l'intégration de cette technologie dans des systèmes complexes (Van der Valk *et al.*, 2020).

Cela marque un tournant dans l'évolution du concept, car, bien que des divergences subsistent, les classifications récentes offrent désormais un socle sur lequel la recherche et les applications futures des jumeaux numériques peuvent se bâtir de manière cohérente et partagée. La taxonomie des jumeaux numériques contribue à clarifier un concept en constante évolution, en offrant des repères structurants malgré la diversité des interprétations sectorielles. Ces efforts de classification facilitent la convergence des définitions et renforcent les bases pour un développement cohérent, adaptable aux multiples contextes d'application.

1.1.4 Jumeau numérique : définition de référence

Enfin, la définition sommative suivante du jumeau numérique est celle à laquelle se conforme l'écriture de ce mémoire :

Un jumeau numérique (*Digital Twin*, DT) est une réplique virtuelle d'un objet, d'un système, ou d'un processus réel, appelé le jumeau réel (*Actual Twin*, AT). Ce dernier alimente en continu le jumeau numérique grâce à des données transmises en temps réel. Il permet une simulation et une analyse détaillée de l'objet, du système ou du processus qu'il représente. Le modèle numérique évolue en fonction des changements observés dans le jumeau réel, et les deux entités, réel et virtuelle, interagissent de manière bidirectionnelle pour permettre une prise de décision en temps réel et une optimisation continue des performances (Mertens, Klikovits, Bordeleau, Denil & Haugen, 2024).

Contrairement à une simple modélisation statique, le jumeau numérique est un système dynamique, alimenté par des données provenant de trois types de sources définies (Chamari *et al.*, 2023a) :

1. ***Devices and systems*** : données provenant des dispositifs IoT, systèmes de gestion de bâtiment (*Building Management System*, BMS), systèmes de ventilation, chauffage, climatisation (*Heating, Ventilation, and Air Conditioning*, HVAC), etc. Ces données sont généralement des séries temporelles recueillies via des capteurs.
2. ***External data sources*** : données externes comme les services météorologiques, les informations sur les utilitaires (énergie, eau), qui sont utilisées pour compléter les informations du bâtiment et anticiper des changements environnementaux par exemple. Ces données sont périodiquement téléchargées, souvent par le biais d'interfaces de programmation d'applications (API) pertinentes.
3. ***Metadata sources*** : données de métadonnées, telles que les modèles BIM, les diagrammes de P&ID (*Piping and Instrumentation Diagram*) [1], et autres sources de données structurées.

[1] Un P&ID est un schéma détaillé représentant les équipements, les conduites, les vannes, les capteurs et les systèmes de contrôle d'un procédé industriel. Il sert de référence essentielle pour la conception,

Ces informations permettent d'ajouter du contexte aux données opérationnelles, facilitant leur gestion et intégration.

Ce modèle virtuel n'est pas simplement une réplique, mais une extension du jumeau réel, capable de prévoir son comportement sous différentes conditions, d'anticiper des problèmes potentiels et d'optimiser ses performances dans un environnement réel. Grâce à des technologies avancées telles que l'IA et la science des données, les jumeaux numériques permettent une gestion proactive des systèmes, en particulièrement précieuse dans des environnements complexes comme les bâtiments intelligents.

Loin d'être figé, le concept de jumeau numérique continue de se préciser à travers des définitions, classifications et modèles qui reflètent sa richesse et son potentiel. Cette pluralité n'est pas une faiblesse, mais une force, témoignant de la capacité du jumeau numérique à s'adapter aux besoins spécifiques de chaque domaine d'application.

1.2 Jumeaux numériques en environnement bâti

Les jumeaux numériques constituent une technologie clé pour les secteurs où les prototypes physiques sont coûteux, où les tests sont difficiles à réaliser, et où la surveillance en temps réel est cruciale (aérospatiale, santé, chaîne d'approvisionnement, etc.). Ils permettent de simuler, optimiser et prédire les performances des systèmes, réduisant ainsi les coûts, les délais et les risques associés (Sharma, Kosasih, Zhang, Brintrup & Calinescu, 2022).

Dans le domaine du bâtiment, les jumeaux numériques peuvent révolutionner la gestion des infrastructures en offrant une modélisation précise, une maintenance prédictive et une optimisation énergétique, tout en facilitant la prise de décision en temps réel, pour des projets plus durables et efficaces.

---

l'exploitation et la maintenance des installations, notamment en ingénierie des procédés (Kang, Lee & Baek, 2019).

Bien que de nombreuses définitions des jumeaux numériques aient été proposées dans la littérature, leur compréhension devient plus complexe lorsqu'il s'agit de les appliquer spécifiquement à l'environnement bâti. L'impact des jumeaux numériques, comme dans d'autres secteurs industriels, est indéniable. Cependant, cette notion demeure complexe et sujette à confusion en raison de l'interaction avec des systèmes connexes tels que le BIM, les systèmes cyber-physiques (CPS) ou encore l'IoT, brouillant ainsi la frontière entre ces concepts (Davari *et al.*, 2022).

1.2.1 Systèmes Cyber-Physiques (CPS)

Davari et al. (2022) explorent précisément ces chevauchements, mettant en lumière les similitudes entre les systèmes cyber-physiques et les jumeaux numériques (Davari *et al.*, 2022). Un CPS désigne un système où des processus physiques interagissent avec des composants computationnels et de communication, permettant un contrôle et une supervision en temps réel grâce aux données collectées par des capteurs et traitées par des algorithmes embarqués (Fritzsch *et al.*, 2023). Il s'agit donc d'un système réactif, capable d'agir directement sur son environnement à l'aide d'actionneurs.

À l'inverse, les jumeaux numériques ne se limitent pas à la régulation en temps réel. Ils proposent une représentation virtuelle évolutive du système physique, intégrant des capacités de simulation, de modélisation et d'analyse prédictive. Là où un CPS assure le fonctionnement opérationnel immédiat (régulation en temps réel des systèmes), un jumeau numérique permet d'anticiper les scénarios futurs et d'optimiser les prises de décision (Davari *et al.*, 2022).

En effet, les CPS et les jumeaux numériques partagent plusieurs caractéristiques communes, notamment l'intégration des données et la correspondance entre les entités physiques et virtuelles. Toutefois, cette comparaison est nuancée par l'introduction de deux sous-catégories : les jumeaux numériques pratiques et les jumeaux numériques idéaux. Tandis que le CPS se rapproche davantage du jumeau numérique pratique, centré sur l'interconnexion des systèmes en temps réel pour le suivi et le contrôle, le jumeau numérique idéal englobe une granularité géométrique

plus complexe et des capacités d'analyse avancées, offrant une modélisation plus fine et une meilleure capacité d'anticipation, ce qui le distingue du CPS (Davari *et al.*, 2022).

1.2.2 Building Information Modeling (BIM)

Le BIM et les jumeaux numériques poursuivent des objectifs convergents, à savoir améliorer la gestion des bâtiments tout au long de leur cycle de vie. Toutefois, leur portée diffère sensiblement. Le BIM est une méthodologie utilisée principalement durant les phases de conception et de construction des infrastructures. Il s'agit d'un modèle numérique détaillant la structure du bâtiment, les matériaux utilisés, ainsi que les informations relatives à l'architecture, à la structure et aux systèmes mécaniques, électriques et de plomberie (*Mechanical, Electrical, and Plumbing*, MEP). Les modèles BIM sont créés à l'aide de logiciels commerciaux tels que Revit [2], qui permettent de générer des représentations 3D des bâtiments en intégrant des informations géométriques et non géométriques, telles que les spécifications, les coûts et les plannings. Cependant, ces modèles restent statiques et ne sont pas conçus pour évoluer en fonction de l'utilisation réelle du bâtiment (Chamari, Petrova & Pauwels, 2023b).

Lorsqu'intégrés dans les phases opérationnelles d'un bâtiment, les modèles BIM peuvent jouer un rôle clé dans la création de jumeaux numériques. En enrichissant le modèle BIM, par une dimension dynamique, il devient possible d'ajouter des données en temps réel provenant de capteurs IoT installés dans le bâtiment. Contrairement au BIM, qui se limite aux informations historiques ou planifiées, le jumeau numérique intègre des données continues, permettant une gestion proactive et une maintenance prédictive, transformant la manière dont les bâtiments sont exploités et maintenus. Ce modèle virtuel va bien au-delà d'une simple réplique ; il s'agit d'un outil intelligent qui évolue selon les conditions réelles et les événements observés dans le bâtiment.

Ainsi, la transition d'un modèle BIM vers un jumeau numérique repose sur l'intégration de technologies avancées telles que le *big data*, l'intelligence artificielle et le *machine learning*

[2] Autodesk Revit : https://www.autodesk.com/products/revit/overview

(ML), renforçant ainsi les capacités d'analyse et d'anticipation. Les capteurs IoT jouent un rôle clé dans cette transformation, en collectant en continu des mesures sur des paramètres critiques tels que la température, l'humidité ou la consommation énergétique. Enrichissant ainsi le modèle BIM pour créer un jumeau numérique qui interagit en permanence avec son environnement physique (Khajavi *et al.*, 2019). Ce processus favorise une exploitation plus efficace des bâtiments, en optimisant leurs performances et en réduisant les coûts opérationnels (Khajavi *et al.*, 2019). Cette transformation est illustrée à la Figure 1.3, qui met en évidence les composants essentiels nécessaires à la création d'un jumeau numérique pour un bâtiment et les différences fondamentales avec le BIM.

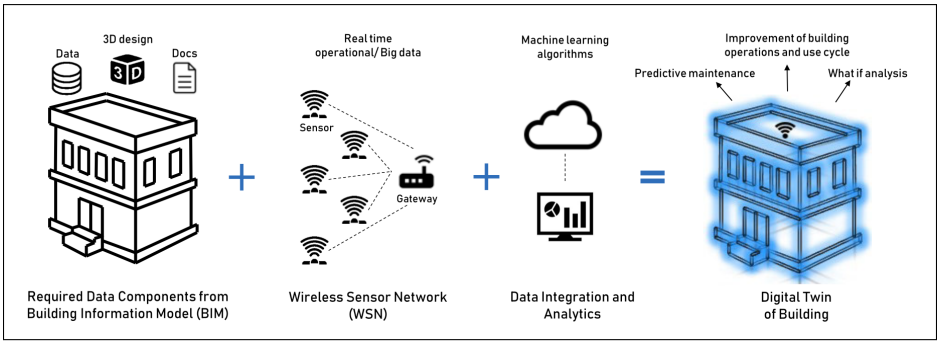


Figure 1.3 Illustration des composants essentiels nécessaires à la création d'un jumeau numérique de bâtiment et des différences avec le BIM
Tirée de Khajavi et al. (2019)

En définitive, alors que le CPS et l'IoT sont orientés vers l'interaction et la gestion en temps réel des systèmes physiques, et que le BIM fournit une base statique décrivant l'infrastructure, seul le jumeau numérique fusionne ces éléments pour offrir une réplique intelligente et évolutive. Grâce à cette interconnexion, il devient un outil stratégique pour la gestion proactive des bâtiments intelligents, intégrant l'analyse en temps réel et l'optimisation continue (Khajavi *et al.*, 2019).

1.3 Enjeux des jumeaux numériques

Le développement de jumeaux numériques dans l'environnement bâti, comme dans d'autres secteurs, soulève de nombreux défis techniques et organisationnels allant bien au-delà de la

gestion des données, bien que celle-ci demeure un enjeu central largement abordé dans la littérature.

Modélisation et conception des jumeaux numériques

L'optimisation du réseau de capteurs dans un bâtiment est un défi important. Il est essentiel de déterminer avec précision le nombre et le type de capteurs nécessaires afin de recueillir des données fiables tout en minimisant les coûts. L'installation optimale des capteurs de manière optimale afin de permettre une couverture complète des différentes zones du bâtiment sans compromettre l'efficacité des dispositifs de mesure, est un processus délicat, nécessitant un calcul minutieux de la configuration de ceux-ci (Khajavi *et al.*, 2019). Par ailleurs, des problèmes d'intégration des données peuvent entraver l'obtention d'une représentation cohérente de l'entité physique et de son environnement, notamment en ce qui concerne le placement optimal des capteurs dans les bâtiments.

L'assurance de la qualité et de la précision des données provenant de divers capteurs et dispositifs est également cruciale. La présence de données inexactes peut entraîner des simulations erronées et la prise de décisions qui s'ensuit (Dihan *et al.*, 2024). Cela nécessite des stratégies avancées pour assurer que les données recueillies soient fiables, précises et continuellement mises à jour (Khajavi *et al.*, 2019). Enfin, la modélisation des objets physiques, indispensable à la création des jumeaux numériques, demeure une tâche complexe qui peut s'appuyer sur des approches expérimentales, mathématiques ou basées sur les données (Khan *et al.*, 2022).

Interopérabilité et intégration des systèmes

L'intégration des données issues de systèmes hétérogènes constitue un défi majeur. Dans le domaine du bâtiment intelligent, plusieurs technologies coexistent, notamment le BMS, l'IoT et le BIM. Chacun de ces systèmes adopte des protocoles et des formats de données variés, ce qui complique leur interopérabilité (Chamari *et al.*, 2023b). Les données opérationnelles (telles que les séries temporelles) et contextuelles (métadonnées) sont souvent stockées dans des formats

distincts et nécessitent des technologies différentes pour leur gestion. Une intégration fluide de ces deux types de données est indispensable pour créer des jumeaux numériques efficaces et exploitables (Chamari *et al.*, 2023b).

Gestion et exploitation des données

L'intégration de multiples types de données (séries temporelles, données multimodales et informations provenant de diverses sources) représente un obstacle majeur dans la mise en œuvre des jumeaux numériques (Sharma *et al.*, 2022). Ce défi est amplifié par l'augmentation exponentielle du volume de données à traiter. Il est donc nécessaire d'adopter des technologies avancées pour fusionner, nettoyer, optimiser et et garantir la qualité des données tout en assurant leur synchronisation en temps réel (Tao *et al.*, 2018). Que ces données proviennent de capteurs IoT, de systèmes de gestion de bâtiments ou d'autres sources externes, leur collecte présente des obstacles considérables à surmonter, notamment en raison de l'ampleur de la diversité des informations à traiter. Le contrôle de l'accès aux données et la sécurisation des informations sont également des enjeux critiques. Les bâtiments connectés via IoT collectent en continu des données sensibles, nécessitant des mécanismes de protection robustes contre les intrusions et les cyberattaques (Shahzad *et al.*, 2022). Ce défi devient particulièrement important dans les environnements où plusieurs parties prenantes (gestionnaires de bâtiments, prestataires de services, occupants) doivent accéder aux données du bâtiment (Khajavi *et al.*, 2019).

Déploiement et architecture des infrastructures

L'architecture informatique sous-jacente aux jumeaux numériques repose généralement sur un modèle distribué combinant le *cloud computing*, l'*edge computing* et des solutions hybrides. Cette distribution permet d'assurer un équilibre entre puissance de calcul et latence, selon les exigences spécifiques des applications. L'edge computing, en particulier, joue un rôle clé en permettant le traitement des données à proximité des capteurs, réduisant ainsi les délais liés aux transferts vers le cloud. Cependant, cette approche pose des défis en matière de synchronisation des données et de gestion des ressources, notamment en raison des capacités limitées des

infrastructures locales (Khajavi *et al.*, 2019). Une solution hybride, combinant *edge* et *cloud computing*, permet d'exploiter les avantages des deux approches. Les traitements nécessitant une faible latence peuvent être réalisés en périphérie, tandis que les analyses plus complexes, impliquant de grandes quantités de données, peuvent être déportées vers le cloud (Khan *et al.*, 2022). La communication entre ces différentes couches d'infrastructure représente également un enjeu majeur. La bande passante disponible et les délais de transmission peuvent affecter la réactivité du jumeau numérique, en particulier dans des applications en temps réel. L'utilisation de réseaux 5G [3] ou de technologies avancées de communication sans fil (Wi-Fi 6, LP-WAN, etc.) peut contribuer à atténuer ces problèmes (Pires, Cachada, Barbosa, Moreira & Leitão, 2019).

Scalabilité et évolutivité des solutions

La gestion de grandes quantités de données des solutions de jumeaux numériques, en particulier dans des environnements complexes où de nombreux capteurs et dispositifs génèrent des informations en continu, représente un facteur clé pour garantir leur pérennité. À mesure que le nombre de capteurs et de dispositifs connectés augmente, il devient essentiel d'adopter des architectures flexibles capables de s'adapter à ces évolutions. Cela inclut la mise en place de systèmes de gestion de bases de données capables de traiter efficacement des données hétérogènes (comme les séries temporelles, les graphes et les métadonnées) et de permettre une analyse rapide et précise. Le défi ici est de garantir que les solutions de données sont suffisamment évolutives pour traiter et intégrer une masse de données toujours croissante, sans compromettre la performance du système. (Chamari *et al.*, 2023b). Les approches distribuées, telles que le traitement en grappes (*cluster computing*), peuvent être employées pour assurer une montée en charge efficace (Chamari *et al.*, 2023b).

Les défis liés au développement des jumeaux numériques dépassent largement la simple gestion des données. Ils englobent des problématiques d'interopérabilité, de sécurité, d'optimisation des infrastructures et de scalabilité. Ces défis, bien que souvent analysés sous l'angle de la gestion des

[3] La 5G est la cinquième génération de réseau mobile, offrant une latence réduite et une bande passante accrue, ce qui est crucial pour les applications en temps réel (Nunna *et al.*, 2015).

données, soulèvent également des questions méthodologiques et organisationnelles. L'évolution des jumeaux numériques, par exemple, ne concerne pas uniquement leur capacité à traiter un volume croissant d'informations, mais aussi leur aptitude à proposer des services toujours plus performants et adaptés aux besoins des utilisateurs. Une approche intégrée, combinant cadre méthodologique, outils et bonnes pratiques, est indispensable pour relever ces défis et exploiter pleinement le potentiel des jumeaux numériques dans l'industrie et l'environnement bâti.

1.4 Outils et architectures pour le développement des jumeaux numériques

Le domaine des jumeaux numériques propose quelques outils et architectures conçus pour faciliter leur développement et leur mise en œuvre, en particulier dans le secteur de l'environnement bâti. Ces solutions, souvent axées sur la résolution de défis spécifiques liés à la gestion des données, ont émergé au fil des années en ce sens. On y retrouve, entre autres, des modèles d'architecture flexibles, des systèmes orientés services, ainsi que des plateformes utilisant des technologies avancées.

1.4.1 Architecture logicielle pour bâtiments intelligents

Ayant d'abord étudié les besoins en données et les exigences spécifiques des bâtiments intelligents, *Brain 4 Buildings (B4B)* [4] a ensuite présenté une architecture de référence pour les bâtiments intelligents. Cette architecture s'inscrit dans une démarche visant à intégrer les données provenant de multiples sources et à faciliter la communication entre elles, notamment en se basant sur une approche modulaire et orientée services. Cette architecture a été mise en œuvre dans un premier temps pour étudier la gestion des données et les exigences spécifiques des bâtiments intelligents. Elle est fondée sur une approche à cinq niveaux, et son but est d'assurer une gestion harmonieuse des données entre différents systèmes et services.

[4] B4B est une initiative de recherche qui vise à exploiter les big data issues des compteurs intelligents, des systèmes de gestion de bâtiments et des dispositifs IoT afin d'optimiser la consommation énergétique, le confort des occupants et les coûts de maintenance. En s'appuyant sur des modèles d'IA et de ML pour adapter la gestion des bâtiments aux comportements des utilisateurs et aux variations énergétiques locales. Détails disponibles sur : https://brains4buildings.org/.

La solution proposée repose sur des microservices, chacun ayant un rôle bien défini dans l'architecture globale. Ces microservices couvrent plusieurs aspects du système, notamment : les sources de données (systèmes d'automatisation des bâtiments, capteurs IoT, services externes), l'intégration et la gestion des données, la logique applicative pour traiter les informations collectées, et la couche de visualisation permettant aux utilisateurs d'interagir avec les systèmes en temps réel. L'une des spécificités de cette architecture est sa flexibilité, permettant d'adapter facilement les composants aux exigences des différents bâtiments et des systèmes impliqués (Chamari *et al.*, 2023a).

Bien que l'architecture ne soit pas spécifiquement dédiée aux jumeaux numériques, elle propose des concepts tels que la classification des types de données (ex. : capteurs IoT, données contextuelles), qui s'appliquent aisément au développement de jumeaux numériques. Cette approche permet non seulement de gérer efficacement les données en temps réel, mais aussi d'assurer une intégration fluide avec des applications tierces, comme celles de maintenance prédictive ou de gestion énergétique, favorisant ainsi la transition vers des bâtiments plus intelligents et durables.

1.4.2 Architecture des bâtiments intelligents comme services : vers une gestion modulaire et intégrée

L'étude de Chamari et al. (2023) propose une architecture orientée services (*Service-Oriented Architecture*, SOA) conçue pour soutenir des applications pilotées par les données dans le domaine des bâtiments intelligents (Chamari *et al.*, 2023b). Cette architecture vise à répondre aux défis liés à l'intégration de sources de données hétérogènes, tout en mettant l'accent sur la modularité et la réutilisabilité des composants. Bien que cette étude ne traite pas explicitement de l'évolution des jumeaux numériques dans le temps, l'architecture proposée jette les bases d'une évolution systématique des JN en fournissant un cadre robuste pour l'intégration et la gestion des données en temps réel (Chamari *et al.*, 2023b).

L'architecture proposée est structurée autour de sept catégories de services, chacune jouant un rôle spécifique dans l'écosystème des bâtiments intelligents :

1. **Applications métier existantes** : Ces applications incluent les systèmes de gestion de bâtiments (BMS), les systèmes de gestion de l'énergie (*Energy Management System*, EMS), les dispositifs IoT, ainsi que des services externes comme les API météorologiques et les services publics. Ces systèmes sont souvent hérités et doivent être intégrés dans l'architecture globale.
2. **Nouvelles applications basées sur des microservices** : Les nouvelles applications sont conçues selon une architecture microservices, ce qui permet une modularité et une extensibilité accrues. Ces services incluent des connecteurs pour les systèmes IoT, des outils ETL (Extract, Transform, Load), des applications pilotées par les données comme le contrôle prédictif de modèle (*Model Predictive Control*, MPC) et la détection de pannes (*Fault Detection and Diagnosis*, FDD), et finalement les services de support pour ces applications pilotées par les données. Ils incluent l'agrégation des données, le nettoyage des données, et les processus ETL, tous essentiels à la préparation et au traitement des données avant leur utilisation dans les applications.
3. **Les bases de données** : L'architecture repose sur plusieurs types de bases de données pour stocker ses différents types de données, notamment des bases de données temporelles pour les séries chronologiques, des bases de données graphiques pour les métadonnées sémantiques, et des bases de données documentaires pour les informations utilisateur et projet.
4. **Logiciels d'intégration** : Ces services facilitent l'intégration des systèmes hétérogènes via des protocoles comme HTTP (*Hypertext Transfer Protocol*) et des systèmes de messagerie tels que MQTT (*Message Queuing Telemetry Transport*), avec des connecteurs spécifiques pour les BMS, IoT, et BIM. Les API permettent d'exposer les services et données aux utilisateurs via des standards comme REST *Representational State Transfer* ou GraphQL (*Graph Query Language*. Elle centralise la sécurité, le contrôle d'accès et la surveillance, évitant leur préocupation au niveau des services distincts.

5. **Services d'infrastructure** : Ces services incluent la conteneurisation via Docker[5] et l'orchestration de conteneurs via Kubernetes[6], qui permettent une gestion efficace des ressources et une scalabilité dynamique des services.
6. **Services partagés** : Ces services communs à toutes les applications incluent la sécurité, la gouvernance, la surveillance et l'automatisation. Ils ne sont pas spécifiques aux bâtiments intelligents mais sont essentiels pour assurer la fiabilité et la maintenabilité de l'architecture.
7. **Interfaces Utilisateur (UI)** : Les interfaces utilisateur, telles que les applications web et mobiles, permettent aux utilisateurs finaux d'interagir avec les données et les services. Ces interfaces incluent des tableaux de bord, des visualisations de données en temps réel, et des outils d'exploration de données.

L'une des contributions majeures de cette architecture est l'intégration des technologies sémantiques pour la gestion des métadonnées. Les ontologies, telles que Brick (*Brick Schema*) et SSN (*Semantic Sensor Network*), sont utilisées pour standardiser la sémantique des données et créer des liens entre les différentes sources de données. Cela permet une intégration plus efficace des données opérationnelles et contextuelles, ce qui est essentiel pour des applications comme les jumeaux numériques (Chamari *et al.*, 2023b).

L'architecture ayant été mise en œuvre dans trois cas d'utilisation distincts, sa modularité et sa réutilisabilité sont ainsi démontrées par les auteurs. Les cas d'utilisation, dans le contexte de bâtiments universitaires (*Atlas campus building* à l'Université de Technologie d'Eindhoven et *Building 33* à l'Université de Technologie de Delft), sont les suivants :

- Application de jumeau numérique : Intégration des données de capteurs avec un modèle BIM pour créer un jumeau numérique d'un bâtiment universitaire.
- Intégration de données en temps réel : Visualisation des données en temps réel provenant de capteurs IoT dans une application web.

[5] Docker est une plateforme permettant de créer, déployer et gérer des conteneurs applicatifs de manière légère et portable. Documentation officielle : https://docs.docker.com/

[6] Kubernetes est un système open-source conçu pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Documentation officielle : https://kubernetes.io/docs/

- Exploration de données BMS : Utilisation de tableaux de bord Grafana [7] pour explorer les données historiques et en temps réel des systèmes BMS, enrichies par des métadonnées sémantiques.

Enfin, l'architecture proposée par Chamari et al. (2023) offre un cadre robuste pour le développement d'applications pilotées par les données dans le domaine des bâtiments intelligents (Chamari *et al.*, 2023b). En mettant l'accent sur la modularité, la réutilisabilité et l'intégration en temps réel, cette architecture pose les bases pour une évolution systématique des jumeaux numériques. Cependant, des lacunes subsistent, notamment en ce qui concerne la génération automatisée de métadonnées, la communication bidirectionnelle, la sécurité des données, et l'interopérabilité avec les systèmes hérités. Les travaux futurs visent à se concentrer sur l'amélioration de ces aspects pour rendre l'architecture plus complète et adaptable aux besoins complexes des bâtiments intelligents (Chamari *et al.*, 2023b).

1.4.3 L'approche DTaaS à travers le cycle de vie des jumeaux numériques

Une architecture de jumeaux numériques vue comme des services transparait dans diverses publications. Talasila et al. (2023) proposent une approche complète en ce sens, articulée autour de la gestion complète du cycle de vie d'un jumeau numérique (Talasila *et al.*, 2023). Cette perspective permet non seulement de simplifier la gestion des actifs numériques mais aussi de garantir leur efficacité tout au long de leur durée de vie, depuis leur création jusqu'à leur mise hors service.

Le DTaaS (Digital Twin as a Service) propose une plateforme logicielle qui traite les jumeaux numériques comme des actifs réutilisables. Quatre catégories d'actifs sont utilisées : données (D), modèles (M), fonctions (F) et outils (T) (Talasila *et al.*, 2023). Cette approche novatrice pour la création, l'exploitation et la gestion des jumeaux numériques à travers une plateforme centralisée, permet aux utilisateurs, qu'ils soient experts ou non, de créer, déployer et gérer

[7] Grafana est une plateforme open-source de visualisation et d'analyse de données, permettant de créer des tableaux de bord interactifs pour surveiller les données en temps réel et historiques. Documentation officielle : https://grafana.com/docs/.

des jumeaux numériques en utilisant des ressources partagées et des actifs réutilisables. L'un des avantages majeurs présenté à travers cette approche est l'intégration de la réutilisabilité des modèles, des données et des outils dans le processus de création des jumeaux numériques. Grâce à cette modularité, les utilisateurs peuvent accéder à une bibliothèque d'actifs prêts à l'emploi, facilitant ainsi l'implémentation rapide de jumeaux numériques sans nécessiter un développement complexe à chaque nouvelle application (Talasila *et al.*, 2023).

La plateforme DTaaS a pour objectif de soutenir l'ensemble du cycle de vie d'un jumeau numérique. Cela inclut la création, l'exécution, l'enregistrement, l'analyse, l'évolution et la fin de vie des jumeaux numériques. La plateforme prend en charge l'intégration de plusieurs types d'actifs tels que des données, des modèles, des fonctions et des outils, permettant une flexibilité et une évolutivité considérables dans le déploiement de ces systèmes. En outre, le modèle repose sur une architecture de microservices qui facilite l'extensibilité, la maintenance et l'intégration avec des services tiers tels que la visualisation des données et la gestion des événements (Talasila *et al.*, 2023). Les utilisateurs de cette plateforme devraient pouvoir bénéficier de fonctionnalités complètes, incluant la création, la consolidation, la configuration, l'exécution, l'exploration, l'enregistrement, l'analyse de scénarios (*what-if analysis*) et le partage des jumeaux numériques. Concrètement, ils ont la possibilité de fournir des actifs de jumeaux numériques (modèles, données, outils, fonctions), un fichier de configuration, et d'obtenir un jumeau numérique basé sur le cloud.

Cependant, bien que cette plateforme représente une avancée importante dans la gestion des jumeaux numériques, plusieurs limitations doivent être prises en compte. L'aspect architecture demeure fragile, notamment en ce qui concerne l'interopérabilité entre les différents modèles et outils présents sur la plateforme. Ayant pour but de simplifier le processus de création et de gestion des jumeaux numériques, elle semble avant tout conçue pour la génération automatique de jumeaux numériques, plutôt que pour un développement personnalisé et évolutif de ceux-ci. L'approche repose sur l'utilisation de composants réutilisables qui, bien que pratiques, peuvent ne pas suffire à répondre aux besoins spécifiques des utilisateurs d'environnements complexes. Enfin, bien que l'intégration avec des services externes soit possible, elle nécessite un effort

important en termes de configuration et de validation des interfaces, ce qui peut compliquer l'utilisation de la plateforme par des utilisateurs non techniques. Ces défis soulignent la nécessité d'une meilleure gestion des configurations et d'une plus grande automatisation dans l'approche DTaaS pour maximiser son efficacité et sa praticité.

1.4.4 Le jumeau numérique cognitif (CDT) : une évolution intelligente

Zheng et al. (2022) présentent le concept de jumeau numérique cognitif (*Cognitive Digital Twin*, CDT), une évolution avancée des jumeaux numériques traditionnels (Zheng *et al.*, 2022). Le CDT se distingue par l'ajout de capacités cognitives, permettant une représentation plus intelligente et autonome des systèmes physiques complexes. Contrairement aux DT classiques, qui se concentrent principalement sur la surveillance et la simulation, les CDT intègrent des technologies intelligentes pour permettre des activités telles que la perception, le raisonnement, la prise de décision autonome et l'apprentissage continu (Zheng *et al.*, 2022).

La couche cognitive au-dessus du JN

Le CDT repose sur une couche cognitive qui nécessite des ajouts technologiques supplémentaires par rapport aux JN traditionnels. Ces ajouts incluent :

- Technologies sémantiques : Les ontologies et les graphes de connaissances permettent de structurer et d'interconnecter les données hétérogènes provenant de différentes sources, facilitant ainsi l'interopérabilité et la création de nouvelles connaissances.
- Intelligence artificielle : Des algorithmes d'IA sont utilisés pour permettre des fonctions cognitives comme la prédiction, la résolution de problèmes et l'optimisation dynamique.
- Gestion du cycle de vie des produits (*Product Lifecycle Management*, PLM) : Le CDT intègre des modèles numériques couvrant toutes les phases du cycle de vie d'un système, de la conception à la fin de vie, en passant par la production et la maintenance.

- Ingénierie des systèmes basée sur les modèles (*Model-Based Systems Engineering*, MBSE) : Cette approche permet de formaliser et de gérer les architectures de systèmes complexes, en alignant les modèles numériques avec les entités physiques.

Architecture de référence du CDT

Zheng et al. (2022) proposent une architecture de référence en trois dimensions pour le CDT, inspirée du modèle RAMI4.0 [8] (*Reference Architecture Model Industrie 4.0*) (Zheng *et al.*, 2022). Cette architecture est conçue pour guider le développement des CDT en intégrant les éléments clés suivants :

1. **Phases du cycle de vie** : Le CDT couvre toutes les phases du cycle de vie d'un système, depuis la conception (BOL - *Beginning of Life*) jusqu'à l'exploitation (MOL - *Middle of Life*) et la fin de vie (EOL - *End of Life*). Cela permet une intégration continue des données et des modèles numériques tout au long du cycle de vie.
2. **Niveaux hiérarchiques du système** : Le CDT est structuré en niveaux hiérarchiques, allant des systèmes complexes (*System of Systems*, SoS) jusqu'aux composants individuels. Cette hiérarchie permet de modéliser et de gérer les interactions entre les différents sous-systèmes et composants.
3. **Couches fonctionnelles** : L'architecture comprend six couches fonctionnelles :
    a. Entités physiques : Représente le système réel dans l'espace physique.
    b. Ingestion et traitement des données : Collecte et traite les données provenant des entités physiques.
    c. Gestion des modèles : Utilise des technologies sémantiques pour stocker et gérer les modèles numériques.

[8] RAMI 4.0 est un cadre de référence standardisé pour l'Industrie 4.0, structurant les systèmes en trois dimensions (hiérarchie, cycle de vie et couches) afin de faciliter l'intégration des technologies numériques dans l'industrie manufacturière. Documentation officielle : https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.html.

d. Gestion des services : Orchestre les services basés sur les données et les modèles pour répondre aux besoins des applications.
e. Gestion des jumeaux : Assure la mise à jour et l'intégration des différents jumeaux numériques.
f. Interaction utilisateur : Permet aux parties prenantes d'interagir avec le CDT via des interfaces frontales.

Le CDT représente une avancée significative par rapport aux DT traditionnels, en ajoutant une couche cognitive qui permet une gestion plus intelligente et autonome des systèmes industriels complexes. L'architecture de référence proposée fournit un cadre structuré pour le développement des CDT, ouvrant ainsi la voie à des applications industrielles plus robustes et adaptatives, alignées sur les objectifs de l'industrie 4.0. Cependant, des défis majeurs subsistent, notamment en termes de gestion des connaissances, d'intégration des modèles, de standardisation et de mise en œuvre pratique. Ces défis devront être surmontés pour pleinement réaliser le potentiel des CDT dans des environnements industriels réels (Zheng *et al.*, 2022).

1.4.5 Architecture de jumeau numérique : cas de l'Université de Cambridge

L'article sur le développement d'un jumeau numérique à l'Université de Cambridge discute de l'architecture d'un JN dynamique à l'échelle des bâtiments, intégrant diverses sources de données hétérogènes, telles qu'un modèle BIM multi-couches et des capteurs IoT (Lu *et al.*, 2020). L'architecture du système est conçue pour soutenir une gestion intelligente des actifs, fournir une gestion efficace des opérations et de la maintenance (*Operation and Maintenance*, O&M), et favoriser l'interaction entre les humains et les bâtiments ou les villes via des canaux visuels et durables.

L'architecture de ce système repose sur plusieurs couches essentielles :

1. **La couche d'acquisition des données** : Elle constitue la base de chaque jumeau numérique. Elle intègre des données provenant de divers capteurs IoT, systèmes de gestion des bâtiments

(BMS), et autres sources, telles que des dispositifs RFID [9] (*Radio-Frequency Identification*) ou des réseaux sans fil. Cette couche est cruciale pour la collecte et la gestion des données provenant d'une variété de sources réparties sur les bâtiments et les infrastructures urbaines.

2. **La couche de transmission des données** : Cette couche est responsable du transfert des données collectées vers les couches supérieures pour la modélisation et l'analyse. Elle utilise des technologies de communication telles que Wi-Fi, Zigbee [10], ou LP-WAN [11] (*Low Power Wide Area Network*), en fonction de la portée et des besoins de transmission des données. L'utilisation de ces technologies permet une gestion à distance des systèmes et assure l'intégration des différentes données dans l'architecture du JN.
3. **La couche de modélisation numérique** : Dans cette couche, des modèles numériques comme le BIM ou le *City Information Modeling* (CIM) sont utilisés pour représenter les actifs physiques et leurs états. Ces modèles servent de base pour simuler et analyser les conditions actuelles des bâtiments et des infrastructures, en prenant en compte les informations environnementales et contextuelles.
4. **La couche d'intégration des données et des modèles** : Cette couche assure l'intégration des données collectées avec les modèles numériques. Elle inclut des fonctions de stockage, d'analyse, et d'intégration de données, soutenues par des technologies telles que l'IA et le ML pour la gestion en temps réel et la prise de décision. Elle facilite l'intégration des données de différents systèmes comme la gestion des actifs et la gestion de la maintenance.
5. **La couche de service** : Enfin, la couche de service permet l'interaction avec les utilisateurs finaux (gestionnaires de bâtiments, utilisateurs) et offre des services basés sur les informations et les analyses issues des couches précédentes. Ces services incluent des fonctionnalités comme la maintenance prédictive, l'optimisation de la consommation énergétique, et la

---

[9] La technologie RFID utilise des ondes radio pour identifier et suivre des objets équipés d'étiquettes RFID, sans nécessiter de contact direct ni de ligne de vue. Elle est couramment utilisée dans la logistique, le contrôle d'accès et les bâtiments intelligents (Jones, Clarke-Hill, Shears, Comfort & Hillier, 2004).

[10] Zigbee est un standard de communication sans fil utilisé pour les réseaux à faible consommation d'énergie, souvent utilisé dans les applications IoT (Sanchez-Iborra, Sanchez-Gomez & Skarmeta, 2018).

[11] LP-WAN (Low Power Wide Area Network) est une technologie de communication sans fil conçue pour les appareils IoT nécessitant une faible consommation d'énergie et une longue portée de transmission (Sanchez-Iborra *et al.*, 2018).

gestion des espaces. Elle permet également de fournir des retours aux utilisateurs pour améliorer l'efficience des systèmes et l'expérience globale.

Cette architecture est illustrée par un démonstrateur à l'échelle du campus de l'Université de Cambridge, qui combine ces couches pour soutenir la gestion des actifs et la maintenance dans un cadre universitaire.

1.4.6 Une base commune pour l'évolution des jumeaux numériques

La notation DarTwin est une approche visuelle développée pour faciliter la compréhension et la gestion de l'évolution des jumeaux numériques. Elle permet de relier les objectifs (pourquoi), les propriétés d'intérêt (quoi) et l'architecture (comment) d'un système de jumeaux numériques, offrant ainsi une vue holistique de son évolution (Mertens *et al.*, 2024).

Les auteurs abordent les défis liés à la création et à la maintenance des jumeaux numériques, en particulier dans un contexte où les systèmes physiques et les besoins des utilisateurs évoluent constamment. Introduisant finalement une notation visuelle permettant de raisonner sur un système de jumeaux numériques à travers ses différentes interrogations ayants de multiples apports.

DarTwin encourage une conception modulaire, facilitant l'ajout, la modification ou le remplacement de composants sans perturber l'ensemble du système, ce qui est essentiel pour intégrer de nouveaux objectifs ou s'adapter aux changements des besoins des utilisateurs. En visualisant les transformations architecturales, DarTwin permet de documenter clairement les ajustements nécessaires, comme l'ajout d'un nouveau JN pour optimiser la consommation d'énergie, tout en montrant ses interactions avec les JN existants. De plus, la notation aide à identifier et résoudre les conflits entre objectifs concurrents, par exemple en introduisant un arbitre pour harmoniser les décisions lorsque le confort thermique entre en tension avec l'économie d'énergie.

Enfin, en reliant objectifs, propriétés d'intérêt et architecture, DarTwin fournit un cadre robuste pour suivre et documenter l'évolution des JN dans des environnements dynamiques, où les

systèmes et les besoins des utilisateurs évoluent fréquemment. Ainsi, DarTwin se positionne comme un outil essentiel pour les ingénieurs et concepteurs de systèmes complexes, favorisant une gestion structurée et évolutive des jumeaux numériques.

En conclusion, bien que de nombreux outils, sous forme d'architectures, de plateformes ou *frameworks* aient été mis en place pour le développement des jumeaux numériques, il existe encore des lacunes, notamment en termes de flexibilité, de scalabilité et d'intégration entre les différents systèmes. Les solutions proposés dans la littérature adressent certains problèmes spécifiques du développement des jumeaux numériques, mais ne couvrent pas l'ensemble du processus. Il est donc essentiel de fournir aux développeurs de jumeaux numériques une base commune qui s'applique à toutes les sphères du développement. Cette fondation permettra de garantir une cohérence et une direction claires dès le départ, tout en offrant la flexibilité nécessaire pour évoluer avec le temps. Ainsi, une telle base contribuera à encadrer et à structurer efficacement le développement des jumeaux numériques dans le contexte des environnements bâtis. C'est ici qu'une approche DevOps pourrait jouer un rôle clé, en apportant des pratiques robustes dès le début du processus de développement. En intégrant des méthodologies DevOps à une architecture bien structurée, il devient possible d'assurer une gestion continue et une évolution fluide des jumeaux numériques. L'application de bonnes pratiques DevOps permet non seulement de structurer efficacement le cycle de vie des applications, mais aussi de garantir la cohérence, l'automatisation et la qualité du développement, tout en répondant aux défis actuels d'intégration et de maintenance des systèmes.

1.5 DevOps

Au cours de la dernière décénie, le DevOps s'est imposé comme l'approche incontournable permettant d'accroître l'agilité, la productivité et la qualité des systèmes dans l'industrie logicielle. Il a gagné en popularité, depuis son émergence au début des années 2010, jusqu'à devenir aujourd'hui un sujet de recherche majeur dans le domaine de l'ingénierie logicielle et de la gestion des systèmes informatiques. Cette évolution a entraîné des avancées remarquables dans divers

aspects essentiels du développement et des opérations logicielles (Forsgren, Humble & Kim, 2018).

Le DevOps est une approche qui combine le développement logiciel (Dev) et les opérations informatiques (Ops) pour améliorer la collaboration, l'automatisation et la livraison continue des applications. Le terme « DevOps » a commencé à émerger à la fin des années 2000, dans un contexte où d'une communauté de praticiens et de chercheurs visant à combler le fossé entre le développement logiciel et les opérations informatiques. La popularisation de ce terme est souvent attribuée à des événements tel que la conférence « DevOpsDays » en 2009, à laquelle des experts des deux domaines se sont réunis pour échanger sur des défis et solutions communes. C'est donc en réponse aux silos organisationnels entre les équipes de développement et d'exploitation, qui entravaient la livraison rapide et fiable des logiciels, que le DevOps est né. Les travaux de Humble & Farley (2010) dans leur livre « Continuous Delivery » ont jeté les bases des pratiques DevOps en mettant l'accent sur l'automatisation, l'intégration continue et la livraison continue (Humble & Farley, 2010).

1.5.1 Fondements et mise en œuvre des pratiques DevOps

Dans leur ouvrage, Kim et al. définissent trois piliers appelés les « trois voies » du DevOps : *Flow*, *Feedback*, et *Continual Learning and Experimentation* (Kim *et al.*, 2021). Ces principes forment un cadre qui guide les équipes dans l'adoption de pratiques DevOps efficaces et permettent de maximiser la performance du développement logiciel et des opérations. Voici un aperçu de chaque voie :

1. ***Flow*** : Le flux se concentre sur l'optimisation du travail, la réduction des temps de cycle et l'amélioration de la livraison continue des applications, ce qui permet de fournir rapidement des fonctionnalités et de la valeur au client. L'objectif est de créer un flux de travail continu et fluide qui minimise les interruptions, les goulots d'étranglement [12] et les délais. Cela se

[12] Un goulot d'étranglement désigne un point du processus où le flux de travail est ralenti, limitant ainsi la performance globale du système.

fait en simplifiant les processus et en automatisant les tâches répétitives, afin de permettre une livraison plus rapide et plus fréquente des logiciels.

2. ***Feedback*** : La rétroaction met l'accent sur l'importance des retours rapides et continus tout au long du cycle de vie du développement. Cela inclut des retours d'informations provenant des tests, des utilisateurs finaux, ainsi que des retours des équipes de développement et des opérations. L'objectif est de détecter les problèmes le plus tôt possible pour les corriger rapidement et d'améliorer constamment les produits. Ce processus est fondamental pour la mise en place de la qualité continue et l'amélioration des pratiques.
3. ***Learning and Experimentation*** : Cette voie encourage une culture d'apprentissage continu et d'expérimentation au sein des équipes. Elle incite les professionnels du DevOps à toujours chercher de nouvelles solutions, à tester de nouvelles idées et à apprendre de leurs erreurs pour améliorer les processus. Cela inclut la mise en place de pratiques comme les rétrospectives, les analyses post-mortem et l'expérimentation contrôlée, permettant ainsi d'identifier de meilleures approches et d'ajuster les méthodes de travail en fonction des retours d'expérience.

Les principes fondamentaux que recouvrent ces trois voient sont appliqué à un workflow en boucle divisé en huit phases : planifier (*plan*), coder (*code*), consolider (*build*), tester (*test*), mise en production (*release*), déployer (*deploy*), exploiter (*operate*), et surveiller (*monitor*). À travers ces phases revenant au début à partir de la fin, de multiples processus sont mis en place afin de répondre aux défis que le DevOps adresse. L'intégration continue (CI) pour l'automatisation des tests et de l'intégration du code, la livraison continue (CD) pour le déploiement automatisé des applications en production et les IaC (*Infrastructure as Code* pour la gestion de l'infrastructure via des scripts et des modèles sans oublier le monitoring et logging pour la surveillance continue des performances et des erreurs, en sont tous des exemples (Bass, Weber & Zhu, 2015).

1.5.2 Apports et défis de l'adoption de DevOps

Le DevOps permet aux organisations de bénéficier de nombreux avantages, parmi lesquels la réduction du *Time-to-Market*, grâce à des cycles de développement accélérés, et l'amélioration

de la qualité logicielle par l'automatisation des tests, réduisant ainsi les erreurs humaines. Cependant, pour tirer pleinement parti de ces bénéfices, il est essentiel d'adresser plusieurs défis majeurs relevés lors de l'adoption de cette approche. Les travaux de (Kim *et al.*, 2021) dans « The DevOps Handbook » mettent en lumière des obstacles fréquents à la mise en œuvre efficace du DevOps, regroupés ici en trois grandes catégories :

1. **Freins culturels et humains** : L'adoption du DevOps implique un changement de culture organisationnelle, orientée vers la collaboration, la transparence et l'expérimentation continue. Cela suppose l'abandon des silos traditionnels entre les équipes de développement, d'opérations et de sécurité. Or, ce changement rencontre souvent des résistances : peur de l'inconnu, réticences à modifier les responsabilités établies, ou encore culture du blâme peu propice à l'innovation. Sans un *leadership* fort et engagé, et sans accompagnement du changement, ces freins humains peuvent ralentir ou compromettre la transformation.
2. **Contraintes organisationnelles et structurelles** : L'absence de vision stratégique claire, le manque de soutien de la direction, ou l'insuffisance des ressources allouées (budget, temps, personnel) limitent souvent la capacité des équipes à adopter pleinement les pratiques DevOps. De plus, les silos fonctionnels persistent dans de nombreuses structures, avec des objectifs et priorités divergents entre équipes, ce qui nuit à la collaboration et à la fluidité des processus. Le manque de formation et de compétences spécifiques constitue également un obstacle fréquent à l'adoption à grande échelle.
3. **Limites techniques et technologiques** : La mise en œuvre de DevOps nécessite un socle technologique robuste. Cela inclut l'automatisation (CI/CD), la surveillance des systèmes (*monitoring* et *logging*), la gestion de l'IaC, ainsi que l'intégration de la sécurité (DevSecOps). Or, de nombreuses organisations se heurtent à des environnements existants complexes ou obsolètes (systèmes hérités monolithiques), à un manque de standardisation des outils, ou encore à une faible visibilité sur les performances. Ces difficultés techniques freinent l'intégration fluide des pratiques DevOps dans les pipelines de développement.

Enfin, ces défis peuvent freiner la transformation DevOps si aucune stratégie d'accompagnement adaptée n'est mise en place.

En résumé, les tendances actuelles incluent, entre autres, l'intégration de la sécurité dans les pipelines DevOps, le GitOps représentant l'utilisation de Git comme source de vérité pour la gestion des infrastructures, ou encore l'utilisation de l'intelligence artificielle et du *machine learning* (AI/ML) pour optimiser les pipelines et prédire les incidents. Il est possible de réaliser des avancées significatives en appliquant les bonnes pratiques du DevOps, et ce à quel que domaine qu'il soit, à condition de bien adresser les défis spécifiques qui y sont associés. Le domaine de la construction n'y échappe d'ailleurs pas, présentant même des enjeux additionnels dus au fait que le développement logiciel n'est pas au cœur du métier traditionnel, ce qui complexifie l'intégration des pratiques DevOps et nécessite une adaptation sur mesure.

Le DevOps est une approche transformative qui a révolutionné la manière dont les organisations développent et déploient des logiciels, comme l'ont démontré les travaux de DORA (*DevOps Research and Assessment*) à travers des analyses empiriques, notamment les rapports annuels « *State of DevOps* ». Ces études ont mis en évidence l'impact significatif des pratiques DevOps sur la performance organisationnelle, en établissant des métriques clés et en fournissant des preuves solides de leur efficacité (Forsgren, Smith, Humble & Frazelle, 2019).

1.5.3 Architecture microservices

L'architecture microservices constitue un atout majeur pour le DevOps, en permettant la décomposition des applications en services indépendants et modulaires. Cela facilite une adoption fluide des pratiques DevOps, notamment en ce qui concerne la livraison continue et l'automatisation des processus de déploiement. En effet, la séparation des fonctionnalités en microservices (composants relativement petits et indépendants), permet une gestion des dépendances plus souple et favorise une scalabilité accrue ce qui améliore la capacité à gérer des charges variables (Abgaz *et al.*, 2023). L'impact sur la rapidité des déploiements est également considérable, car chaque microservice peut être développé, testé et déployé indépendamment

des autres, ce qui réduit les risques de couplage et permet une mise en production plus rapide et plus fréquente (GitLab, 2022).

L'architecture microservices complète donc parfaitement les principes du DevOps en apportant des avantages directs aux pipelines CI/CD, qui bénéficient de la modularité et de la flexibilité qu'offre une telle architecture (GitLab, 2022). En divisant les applications en petites unités autonomes, elle permet non seulement de minimiser les risques associés aux déploiements, mais aussi d'améliorer la résilience des systèmes grâce à une gestion décentralisée des services (Abgaz *et al.*, 2023). Cette décentralisation favorise aussi l'agilité, permettant aux équipes de déployer rapidement de nouvelles fonctionnalités tout en maintenant une qualité élevée. Ainsi, l'intégration des microservices, organisés autour de fonctionnalités métier spécifiques, dans un environnement DevOps constitue une véritable clé pour améliorer l'efficacité, l'agilité et la fiabilité des déploiements logiciels. Elle favorise l'alignement entre les équipes de développement et les besoins de l'entreprise, tout en permettant une évolution technologique plus fluide.

Cependant, bien que les microservices offrent de nombreux avantages, ils ne sont pas sans défis. Selon Abgaz et al. (2023), la décomposition des monolithes en microservices demeure une tâche complexe et coûteuse, nécessitant une gestion minutieuse des dépendances et une orchestration efficace des services (Abgaz *et al.*, 2023). La communication entre les microservices, par exemple, peut générer des latences et ajouter des points de défaillance supplémentaires. En outre, la charge opérationnelle croît, car chaque microservice doit être surveillé, maintenu et mis à l'échelle de manière indépendante, ce qui requiert des outils et des compétences spécifiques (Abgaz *et al.*, 2023).

1.5.4 Avancées du DevOps pour les jumeaux numériques

Le DevOps, tout comme les jumeaux numériques, est au cœur des évolutions technologiques de ces dernières années. Les avancées dans ces domaines commencent à être abordées de manière commune, avec des publications explorant les synergies possibles entre ces deux approches.

Le DevOps est déjà appliqué à plusieurs systèmes connexes aux jumeaux numériques, tels que ceux mentionnés précédemment. Parmi eux, les CPS sont les plus concernés, mais des approches DevOps existent également pour l'IoT. En revanche, les considérations spécifiques du DevOps appliqué aux jumeaux numériques restent encore rares.

Le DevOps pour les CPS est abordé sous différents angles, notamment à travers l'adoption d'architectures en microservices, l'automatisation des processus grâce à l'IA, ou encore l'intégration des principes du DevOps et des jumeaux numériques pour concevoir des *frameworks* et des plateformes basés sur ces architectures, répondant ainsi aux besoins des CPS.

Outils DevOps pour le CPS

Eramo et al. (2021) présentent *AIDOaRt*, un cadre DevOps enrichi par l'intelligence artificielle pour les systèmes cyber-physiques (Eramo *et al.*, 2021). Ce *framework* repose sur l'ingénierie dirigée par les modèles (*Model-Driven Engineering*, MDE) pour améliorer la conception, le codage, les tests et la surveillance des CPS. Il intègre des techniques d'IA, notamment l'apprentissage automatique (ML), pour automatiser les décisions et les tâches de développement. L'apport principal est une plateforme qui combine des outils DevOps existants avec des méthodes basées sur l'IA, permettant une amélioration continue de la productivité, de la qualité et de la prévisibilité des CPS. Ce cadre est conçu pour être utilisé dans des cas industriels complexes, offrant une solution reproductible et adaptable pour le développement continu de systèmes critiques. Il existe tout de même des défis et des pratiques associés à l'adoption des microservices et de DevOps dans les CPS, en particulier lors de la migration des systèmes existants. On retrouve, les défis relevés, l'hétérogénéité technologique des dispositifs, les exigences de traitement en temps réel, la gestion de grandes quantités de données, ainsi que des défis organisationnels comme l'adaptation des processus CPS à une culture DevOps. En outre, la complexité de la modularisation des systèmes *legacy* et le manque de personnel expérimenté avec les technologies modernes sont également des obstacles fréquemment mentionnés (Fritzsch *et al.*, 2023).

Hugues et al. (2020) proposent une approche similaire intitulée TwinOps, qui fusionne également DevOps et l'ingénierie basée sur les modèles pour améliorer l'intégration et les tests des CPS (Hugues, Hristosov, Hudak & Yankel). L'approche utilise des modèles précis pour la validation et la vérification des logiciels, remplaçant les abstractions simplistes par des simulations détaillées. TwinOps introduit, en plus, une architecture qui automatise les pipelines CI/CD pour générer du code à partir de modèles, permettant ainsi de simuler des systèmes complexes avant leur déploiement. L'apport clé est la création de jumeaux numériques, qui permettent de comparer les systèmes réels avec leurs simulations, améliorant ainsi la validation des modèles et réduisant les erreurs de conception. Cette approche est particulièrement utile pour les systèmes critiques où la précision et la fiabilité sont essentielles.

DevOps pour l'IoT

López-Peña et al. (2020) présentent une approche DevOps pour les systèmes IoT, centrée sur la surveillance rapide et continue de la disponibilité des systèmes (López-Peña, Díaz, Pérez & Humanes, 2020). L'approche est formalisée à travers une activité appelée « *F&CF availability* » (*Fast and Continuous Feedback availability*), qui utilise le modèle SPEM 2.0 [13] (*Software Systems Process Engineering Metamodel*) pour définir des tâches de surveillance, d'analyse et de rétroaction. L'apport principal est l'introduction du concept de *Monitoring as Code* (MaC), une extension de l'IaC, qui permet aux équipes DevOps de configurer des métriques et des indicateurs de surveillance en temps réel. Cette plateforme de surveillance est entièrement automatisée, versionnée et reproductible, ce qui facilite la détection précoce des anomalies et des pannes dans les systèmes IoT distribués. L'approche est validée dans un scénario réel de gestion de l'eau potable, démontrant son efficacité pour améliorer la disponibilité et la fiabilité des systèmes IoT. Elle améliore l'intégration et les tests des systèmes cyber-physiques, tout en relevant les

[13] Le Software Systems Process Engineering Metamodel (SPEM) 2.0 est une norme de l'OMG (Object Management Group) permettant de modéliser et de documenter les processus de développement de logiciels et de systèmes. Il est utilisé pour structurer et formaliser des méthodologies comme celles appliquées aux systèmes IoT (Bendraou, Combemale, Crégut & Gervais, 2007). Voir : https://www.omg.org/spec/SPEM/2.0/.

défis liés à la complexité des systèmes, à la détection tardive des incohérences, à la validation des modèles et à l'adaptation des pratiques DevOps aux environnements multi-domaines.

DevOps pour les jumeaux numériques

Il existe finalement très peu d'études réalisées impliquant l'utilisation du DevOps pour le développement des jumeaux numériques, bien que certaines pratiques aient parfois été mises en œuvre. L'architecture microservices appliquée aux plateformes, présentée dans la section des outils pour les jumeaux numériques, en est un exemple. Cependant, cela ne suffit pas à considérer qu'il s'agit de DevOps. Un cadre DevOps intégré pour une plateforme de jumeaux numériques en tant que service (DTaaS) est tout de même présenté par Scherma (2014) dans le cadre de sa maîtrise en ingénierie informatique (Scherma, 2024). L'approche repose sur l'automatisation des pipelines CI/CD via GitLab, permettant aux utilisateurs de gérer le cycle de vie des jumeaux numériques de manière fluide. L'apport principal est une architecture qui intègre Gitbeaker, une bibliothèque client pour Node.js, permettant aux utilisateurs d'interagir avec l'API de GitLab, facilitant ainsi la création, l'exécution et la surveillance des jumeaux numériques. La plateforme propose également des interfaces utilisateur intuitives, conçues pour simplifier la gestion des jumeaux numériques, même pour les utilisateurs non techniques. Ce cadre permet une collaboration efficace entre les utilisateurs, en leur offrant des espaces de travail privés et une bibliothèque partagée d'actifs. L'approche est validée par des tests utilisateurs, confirmant son efficacité pour simplifier la gestion des DT et améliorer l'expérience utilisateur.

En dépit des nombreuses avancées technologiques et des architectures proposées pour le développement des jumeaux numériques, des défis importants subsistent, notamment en ce qui concerne l'intégration des systèmes, la gestion des données hétérogènes et l'automatisation des processus. Les solutions existantes, bien que prometteuses, ne couvrent pas tous les aspects du cycle de vie des jumeaux numériques, particulièrement dans des environnements complexes comme les bâtiments intelligents. Une base commune pour l'évolution des jumeaux numériques, prenant en compte ces différents défis, est nécessaire pour garantir une évolution fluide et une

gestion efficace des systèmes. Dans ce contexte, l'intégration de pratiques DevOps, avec un accent particulier sur l'automatisation des processus et l'amélioration continue, pourrait jouer un rôle clé pour surmonter ces défis, en permettant une gestion harmonieuse du cycle de vie des jumeaux numériques. Ces avancées ouvriront la voie à une meilleure interopérabilité, une plus grande flexibilité et une performance accrue des jumeaux numériques, particulièrement dans des secteurs où l'efficacité et la réactivité sont cruciales. De plus, alors que les systèmes évoluent continuellement, notamment avec l'intégration croissante de l'intelligence artificielle et de l'Internet des objets, les jumeaux numériques devront s'adapter et évoluer de manière dynamique pour répondre aux besoins changeants et complexes des environnements industriels. Cette capacité d'adaptation continue sera essentielle pour maintenir leur pertinence et leur efficacité à long terme.

CHAPITRE 2

MÉTHODOLOGIE

Le cadre de cette recherche s'articule autour du jumeau numérique du laboratoire du GRIDD à l'ÉTS, qui constitue le principal terrain d'étude et d'expérimentation. Ce laboratoire est dédié à la transformation numérique, à l'industrialisation, la collaboration et la durabilité dans le secteur de la construction. Les membres de ce laboratoire, accompagnés des professeurs encadrants, y conduisent divers projets de recherche et d'innovation. C'est en s'appuyant sur ces développements, en interrogeant les auteurs de travaux pertinents et à travers des observations directes que l'approche proposée dans ce document a été élaborée.

2.1 Le GRIDD : terrain d'étude et d'innovation

Le GRIDD se consacre à l'innovation dans le domaine de la numérisation des bâtiments intelligents. Il s'inscrit dans une perspective de transformation numérique, visant à optimiser la gestion et l'exploitation des infrastructures bâties à l'aide de technologies avancées, notamment les jumeaux numériques.

Le laboratoire du GRIDD, situé au sein même de l'ÉTS, constitue un espace de travail multifonctionnel dédié à la recherche et à l'enseignement. Il sert de terrain d'expérimentation pour tester et valider des approches innovantes en matière de modélisation et de gestion des bâtiments. Dans ce cadre, un modèle BIM du laboratoire a été créé, servant de base à une démarche exploratoire ayant donné lieu à une multitude de projets appliqués, dont le développement progressif de son propre jumeau numérique. Ce contexte a ainsi offert un environnement idéal pour observer, expérimenter et affiner des pratiques de développement adaptées aux jumeaux numériques dans les environnements bâtis.

2.1.1 Jumeau numérique pour la gestion du confort des occupants

Une solution innovante pour la surveillance en temps réel du confort des occupants dans les bâtiments, basée sur des capteurs IoT *plug-and-play* et appuyée par le développement de

jumeaux numériques, a été élaborée. Le confort des occupants est influencé par plusieurs facteurs environnementaux, tels que la température, l'humidité, la qualité de l'air et le niveau sonore. Cependant, évaluer ces paramètres en temps réel est souvent complexe et coûteux (Merabet *et al.*, 2021). La recherche visait à combler cette lacune en proposant une solution intégrée qui permettrait aux gestionnaires de bâtiments de surveiller et d'optimiser le confort des occupants tout en réduisant la consommation d'énergie (Sharafdin, 2024).

Un réseau de capteurs basé sur la technologie *Bluetooth Low Energy* (BLE) a été développé pour mesurer les variables environnementales (température, humidité, qualité de l'air, niveau sonore, etc.). Ces capteurs sont conçus pour être facilement installés et intégrés dans différents environnements bâtis, sans nécessiter de modifications majeures des systèmes existants.

Les données collectées par les capteurs ont ensuite été transmises en temps réel à une plateforme cloud (Microsoft Azure) pour le stockage et l'analyse. Cette intégration permet de gérer de grandes quantités de données et de les visualiser de manière efficace.

Un jumeau numérique de l'espace a finalement été créé en intégrant les données des capteurs IoT avec le modèle BIM. Ce jumeau numérique permet de visualiser en temps réel les conditions de confort dans chaque pièce du GRIDD. En outre, un tableau de bord interactif a été développé pour offrir aux gestionnaires des installations une interface ergonomique permettant de surveiller et d'optimiser les conditions de confort en temps réel. Ce tableau de bord est directement intégré au jumeau numérique, comme présenté à la Figure 2.1, assurant ainsi une gestion proactive des conditions environnementales. L'efficacité de la solution a été évaluée à travers des entretiens avec des gestionnaires d'installations, qui ont confirmé son utilité et son adaptabilité à différents types de bâtiments (Sharafdin, 2024).

2.1.2 Gestion du confort des occupants en constante évolution

Le jumeau numérique du GRIDD a par la suite été au centre de plusieurs projets, notamment des projets de fin d'études (PFE) dans le domaine du génie logiciel à l'ÉTS, explorant différentes approches pour enrichir et optimiser ses fonctionnalités. Le premier PFE visait à créer un outil

Figure 2.1 Intégration du tableau de bord de suivi du confort au jumeau numérique du GRIDD
Tirée de Sharafdin (2024)

de visualisation et de simulation des aspects de gestion du confort des occupants du laboratoire, toujours en s'appuiant sur des données provenant de diverses sources, notamment le modèle BIM et des capteurs installés dans le GRIDD.

Le projet a été mené en suivant une approche Agile [1] et DevOps, garantissant une organisation structurée, une résilience accrue et une maintenance facilitée du système. L'utilisation d'un tableau Kanban [2] pour la gestion des tâches, combinée à des réunions hebdomadaires, a permis de suivre l'avancement du projet et d'assurer une progression itérative et incrémentale en alignement avec les objectifs définis.

Le projet a abouti à la création d'un jumeau numérique fonctionnel, capable de visualiser les données des capteurs, comme illustré à la Figure 2.2 (Karamat, Lahlali & Mercier, 2023).

[1] L'approche *Agile* est un ensemble de principes et de méthodologies visant à rendre le développement logiciel plus flexible et adaptatif. Elle repose sur des cycles de développement courts, appelés *itérations*, favorisant l'amélioration continue et l'implication des parties prenantes (Manifesto, 2001).

[2] *Kanban* est une méthode de gestion du travail issue du système de production de Toyota, utilisée en développement logiciel pour visualiser les tâches, limiter le travail en cours et améliorer l'efficacité du flux de production (Anderson, 2010).

Figure 2.2 Visualisation des capteurs dans Unreal Engine
Tirée de Karamat et al. (2023)

2.1.3 Gestion et optimisation des espaces physiques

Le projet qui s'en est suivi s'inscrit dans une tendance croissante de l'utilisation des technologies numériques pour améliorer la gestion et l'optimisation des espaces physiques. Il repose sur des technologies de pointe en matière de reconnaissance d'images, de modélisation 3D et de gestion de données en temps réel, tout en intégrant également des pratiques DevOps et Agile pour expérimenter de nouvelles technologies et identifier les solutions les plus adaptées au développement d'un jumeau numérique optimal pour le GRIDD. L'objectif principal est de surveiller en temps réel l'inventaire des objets dans la salle et de suivre leur position spatiale grâce à une représentation virtuelle en 3D. Ce jumeau numérique permet de visualiser les objets détectés par une caméra et de les positionner dans un environnement 3D modélisé avec Unity.

La reconnaissance d'images s'effectue à l'aide d'un module `NVIDIA Jetson Xavier NX`, qui traite en temps réel les flux captés par une caméra *Intel RealSense D435*, permettant ainsi d'identifier et de classifier les objets présents dans la salle (Figure 2.3). Une fois détectés, ces objets sont automatiquement ajoutés à l'inventaire numérique et intégrés à un modèle 3D qui

reproduit fidèlement l'espace physique en temps réel. Une interface utilisateur a également été développée afin de faciliter la visualisation et la gestion des objets détectés au sein d'un tableau de bord interactif, tel qu'illustré à la Figure 2.4, permettant aux gestionnaires de suivre en temps réel l'occupation des espaces et l'inventaire des équipements disponibles.

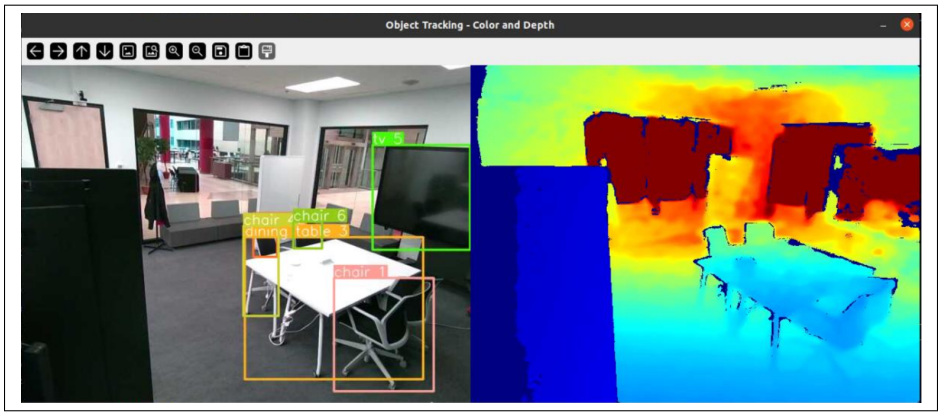


Figure 2.3 Résultats de la détection d'objets traités par le NVIDIA Jetson Tirée de Darnaud et al. (2024)

Figure 2.4 Interface utilisateur de gestion de l'inventaire Tirée de Darnaud et al. (2024)

2.1.4 Expérimentation de JuNo-OPS sur le jumeau numérique du GRIDD

Finalement, un nouveau projet de fin d'études est en cours sur le jumeau numérique du GRIDD, avec pour objectif d'enrichir les fonctionnalités existantes et d'explorer de nouvelles possibilités d'évolution. Contrairement aux projets précédents, celui-ci adopte dès son lancement l'infrastructure proposée en s'appuiant sur le *framework* JuNo-OPS, garantissant ainsi l'application des bonnes pratiques DevOps dès les premières phases de développement, et ce, sous une supervision étroite assurée par des rencontres hebdomadaires avec l'équipe de développement.

Cette approche permet non seulement de structurer efficacement le développement, mais aussi d'évaluer l'impact réel de JuNo-OPS sur la gestion et l'évolution des jumeaux numériques en environnement bâti. En s'appuyant sur l'architecture proposée, l'infrastructure mise en place pour organiser les développements (notamment la gestion des projets sur GitLab [3] via des groupes [4] et la structuration des dépôts [5]), ainsi que sur des pipelines d'intégration et de déploiement continus, le projet suit une méthodologie alignée sur les bonnes pratiques DevOps et un développement Agile. Cette organisation optimise l'automatisation des processus, la scalabilité des services et la collaboration entre les différentes parties prenantes, garantissant ainsi une évolution fluide et maitrisée du jumeau numérique.

Les résultats obtenus dans le cadre de ce projet seront suivis et analysés afin d'en tirer des conclusions sur l'efficacité et la pertinence de JuNo-OPS en tant que cadre méthodologique pour le développement et l'évolution des jumeaux numériques. Cette expérimentation en conditions réelles servira de base pour affiner les principes du *framework* et adapter ses mécanismes d'intégration aux défis concrets rencontrés dans le secteur du bâti.

---

[3] GitLab est une plateforme de développement et de gestion de code source *open source*, conçue pour les projets DevOps et DevSecOps à grande échelle. Elle intègre des fonctionnalités de gestion de référentiels, d'intégration et de déploiement continus (CI/CD) ainsi que des outils de collaboration et de sécurité. Voir : https://about.gitlab.com/.

[4] Un *groupe*, dans GitLab, permet d'organiser plusieurs dépôts sous une même gestion, facilitant ainsi la collaboration, la gestion des permissions et l'attribution des rôles au sein des équipes de développement.

[5] Le terme *dépôt* est l'équivalent français du terme anglais *repository* (souvent abrégé en `repo` dans la communauté des développeurs). Il désigne un espace de stockage utilisé pour la gestion des versions de code, notamment sur Git.

2.2 Analyse exploratoire et élaboration du cadre

L'analyse des projets de jumeaux numériques développés au sein du GRIDD, combinée aux entretiens menés auprès des acteurs impliqués, a mis en évidence un constat unanime : le développement des jumeaux numériques dans les environnements bâtis souffre d'un manque de structuration et de bonnes pratiques.

Afin de mieux comprendre les pratiques actuelles et les défis concrets rencontrés dans le développement de ces systèmes, des entretiens semi-structurés ont été réalisés au cours de la phase exploratoire de cette recherche. Ces échanges ont été menés entre septembre et décembre 2023 auprès de différents profils impliqués dans les projets du GRIDD, ainsi qu'auprès de collaborateurs issus d'autres institutions. Ils incluent des étudiants à la maîtrise et au doctorat, des finissants en génie logiciel travaillant sur des projets de fin d'études, ainsi que des discussions informelles avec les directeurs de recherche et leurs collaborateurs.

Les entretiens, d'une durée variable, ont pris la forme de discussions semi-structurées ou informelles, orientées par une grille de questions visant à comprendre les tâches réalisées, les difficultés rencontrées au quotidien et les limites dans l'évolution des projets (voir le guide d'entretien à l'Annexe I, Tableau I-1). Les réponses ont été prises en notes et synthétisées afin d'identifier les problématiques récurrentes, sans associer directement les constats à un projet spécifique ou à un individu.

Une synthèse des profils interrogés, de leur rôle, de leur implication dans les projets et de la méthode de recueil utilisée est également présentée à l'Annexe I, Tableau I-2.

Ces échanges, nourris par la diversité des projets menés au GRIDD, qu'il s'agisse de bâtiments intelligents ou d'autres types d'ouvrages complexes, ont permis de recueillir une pluralité de points de vue et d'expériences par les chercheurs et développeurs impliqués dans ces initiatives. Ce processus s'est révélé concluant car bien que les problématiques soulevées soient nombreuses et parfois spécifiques à chaque projet, elles convergent vers un besoin commun, à savoir la

nécessitée d'un cadre méthodologique structuré et de bonnes pratiques du génie logiciel adaptées à ce domaine.

L'analyse transversale des entretiens a permis de faire émerger plusieurs défis récurrents, présentés ci-dessous.

Absence de documentation structurée

La documentation a été identifiée comme un défi majeur dans le développement des jumeaux numériques. Plusieurs projets en cours souffrent d'un manque de documentation structurée, rendant leur intégration et leur évolution difficiles. Un cas frappant concerne un jumeau numérique développé pour une infrastructure complexe, comportant plusieurs modules fonctionnels destinés à différents aspects du système. L'absence de documentation initiale pour chacun de ces modules, qui ont été pris en charge par différents intervenants au fil du temps, a rendu l'intégration globale du système laborieuse. La tâche de structurer cette documentation a ainsi reposé a posteriori sur une seule personne, représentant une charge considérable et un défi majeur pour reconstituer la logique d'ensemble du système. Cette situation illustre les difficultés engendrées par un manque de vision documentaire dès les premières phases du développement.

Le manque de découplage entre les différentes couches de développement

Un autre défi récurrent est l'absence de séparation claire entre les différentes couches du développement, ce qui entraîne une charge de travail excessive et une dispersion des compétences. Lors des entretiens, des développeurs *back-end*, travaillant principalement avec le langage `Python`, ont souligné la difficulté d'avoir à intervenir sur des fonctionnalités *front-end* utilisant la bibliothèque `React`, bien que cela ne relève ni de leur expertise ni de leur champ d'intervention habituel. Cela complexifie le travail des équipes et ralentit le développement, car ces développeurs doivent non seulement gérer leur propre domaine et aller combler leur savoir en cherchant à maitriser des technologies qui leur sont étrangère, mais aussi se familiariser avec des technologies qui ne relève en rien à leur mission première. Cette situation illustre un couplage excessif entre

les composantes du système, empêchant une spécialisation efficace et augmentant le risque d'erreurs. Une meilleure modularité, soutenue par une architecture segmentée et des pratiques DevOps adaptées, permettrait aux développeurs de se concentrer sur leur périmètre d'expertise sans être contraints de gérer des évolutions relevant d'autres domaines, optimisant ainsi la productivité et la maintenabilité des applications.

Évolution et cloisonnement des équipes

Les difficultés liées à l'évolution des projets de jumeaux numériques ne se limitent pas aux aspects techniques, mais concernent également l'organisation des équipes. Un cas emblématique concerne un projet antérieur visant à intégrer des données aux systèmes de jumeaux numériques. Bien que fonctionnel à sa mise en place, ce projet a rencontré de sérieuses difficultés d'évolution en raison de problèmes de communication entre les parties prenantes. L'ajout de nouveaux cas d'usage s'est révélé inefficace, mettant en lumière un cloisonnement excessif des équipes et une séparation des responsabilités mal définie.

Ce développement en silos a été un point récurrent des entretiens. De nombreux acteurs ont exprimé leur difficulté à appréhender l'ensemble du processus, se concentrant exclusivement sur leur tâche spécifique. Par exemple, un spécialiste du traitement des données environnementales peut optimiser le stockage et l'analyse des volumes de données sans prendre en compte leur circulation au sein du système ni les contraintes qu'elles peuvent engendrer pour d'autres équipes. Cette approche fragmentée, bien que parfois efficace à court terme, complique la coordination et l'évolution des jumeaux numériques à long terme.

La gestion des environnements de développement

Les aspects liés à la gestion des environnements de développement ont également été identifiés comme un frein au développement des jumeaux numériques. La conteneurisation, pratique courante en DevOps, répond à des défis techniques tels que la gestion des bibliothèques et des dépendances, en particulier pour les développeurs utilisant Git comme source unique de vérité

pour le code. Lors des entretiens, plusieurs professionnels de la construction ont souligné la complexité de la gestion des dépendances et l'impact négatif d'un manque de standardisation sur la maintenance et l'évolution des jumeaux numériques.

Limites des outils existants et contraintes technologiques

En plus des défis organisationnels et techniques, certaines limitations inhérentes aux outils existants ont été mises en avant. Par exemple, Autodesk [6], outil largement utilisé dans la modélisation BIM, ne prend pas en charge la documentation intégrée, ce qui représente un obstacle pour la gestion efficace des connaissances. Bien que les bonnes pratiques DevOps ne puissent pas répondre directement à ce type de contrainte technologique, un cadre structurant accompagné d'outils variés inspirés de cette approche pourrait permettre d'élaborer des solutions évolutives pour pallier ces lacunes à l'avenir.

L'ensemble de ces observations met en évidence la pertinence d'un cadre méthodologique inspiré du DevOps pour structurer et encadrer le développement des jumeaux numériques d'environnement bâtis. En intégrant des pratiques telles que le développement itératif, la gestion collaborative des environnements, la séparation des responsabilités et l'automatisation des processus, il serait possible d'améliorer significativement la cohérence et la pérennité de ces systèmes.

Ces constats ont ainsi renforcé l'hypothèse selon laquelle le DevOps pourrait offrir une réponse adaptée aux défis inhérents au développement des jumeaux numériques dans le secteur de la construction. Cette orientation a conduit à approfondir les recherches sur des sujets connexes, afin d'éclairer les enjeux et les opportunités qu'implique l'application de cette méthodologie à la numérisation des environnements bâtis.

[6] Autodesk est une entreprise spécialisée dans les logiciels de conception et d'ingénierie, notamment utilisée dans le domaine de la modélisation BIM. Voir : https://www.autodesk.com/

2.2.1 Orientations issues des constats

Les constats tirés de l'analyse des projets réalisés au sein du GRIDD, ainsi que des entretiens menés avec les parties prenantes ont permis de faire émerger des problématiques concrètes liées au développement de jumeaux numériques pour les environnements bâtis. Ces observations ont mis en lumière les principaux besoins méthodologiques auxquels cette recherche souhaite répondre. Elles ont ainsi orienté le développement d'une approche adaptée, centrée sur la mise en place d'un cadre structurant pour accompagner le développement et l'évolution de ces systèmes complexes.

Les besoins identifiés ont conduit à la formulation d'une solution reposant sur trois axes complémentaires :

- **Une méthodologie logicielle** pour encadrer le développement systématique et itératif des jumeaux numériques, favorisant une meilleure planification, traçabilité et maintenabilité des solutions mises en œuvre.
- **Une architecture en modulaire**, permettant un découplage fonctionnel des composantes du système, l'indépendance des développements et une intégration simplifiée des contributions logicielles et métier. Elle repose sur des mécanismes d'abstraction des flux de données, qui permettent aux contributeurs non techniques de se concentrer sur le développement d'aspects fonctionnels sans avoir à gérer les dépendances logicielles sous-jacentes.
- **Une infrastructure DevOps**, soutenant l'intégration et le déploiement continus (CI/CD), la gestion des environnements de développement, l'automatisation des processus et la cohérence des pratiques à travers les principales phases du développement logiciel, avec des effets bénéfiques sur l'ensemble du cycle de vie.

Le présent travail s'inscrit ainsi dans cette dynamique, en posant les bases d'un cadre unifié, le *framework* JuNo-OPS, dont la mise en œuvre vise à structurer et outiller le développement des jumeaux numériques dans un contexte réel, tout en facilitant leur évolution continue et en assurant leur maintenabilité à long terme.

CHAPITRE 3

JUNO-OPS : A COMPREHENSIVE DEVOPS FRAMEWORK FOR THE SYSTEMATIC ENGINEERING AND EVOLUTION OF DIGITAL TWINS FOR BUILT ASSETS

Sara Aissat[1] , Jonathan Beaulieu[1] , Francis Bordeleau[1] , Érik Poirier[2] , Ali Motamedi[2] , Julien Gascon-Samson[1]

[1] Département de génie logiciel et des TI, École de Technologie Supérieure
[2] Département de génie de la construction, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Article soumis à la revue « Software and Systems Modeling » (SoSyM), décembre 2024.

Abstract

Digital Twins (DT) constitute complex software systems that need to be continuously modified/updated to meet evolving user requirements and priorities, and support continual improvement. Because they aim at monitoring and improving different system aspects, their development requires the collaboration of people from different domains of expertise, e.g. the development of a DT for built assets may involve the collaboration of experts in software engineering, thermal comfort, air quality, energy consumption, etc. Consequently, DTs need to be engineered to enable the fast and secure integration and deployment of new code, the systematic and iterative evolution of their components, and the independent development of different system aspects by those experts.

In this paper, we present JuNo-OPS, a DevOps framework for the engineering of DTs for built assets. The framework is being developed, tested and validated in the the context of various DT projects at École de Technologie Supérieure (ÉTS), including the development of a DT for a multi-functional room. We focus on two main aspects of DT engineering : the DT architecture ; and the DevOps infrastructure used to support the development, continuous integration, and continuous deployment of the DT software. We also discuss challenges and next steps related to the development and evolution of the framework.

Keywords : DevOps, Digital Twins, CI/CD Pipelines, Microservices, IoT

3.1 Introduction

Digital Twin (DT) emerged over the last decade as a key technology at the core of the digital transformation to improve and optimize various aspects of objects, systems, or processes [1]. A DT is a digital representation of an actual system [2], referred to as the Actual Twin (AT), that is dynamically and constantly updated with system data, and provides a set of services related to its behavior and environment (Mertens, Klikovits, Bordeleau, Denil & Haugen, Accepted for publication, 2024). DTs are now used in a broad range of application domains, including aerospace (Li, Aslam, Wileman & Perinpanayagam, 2021), agriculture (Purcell & Neubauer, 2023), automotive (Piromalis & Kantaros, 2022), construction (Akanmu, Anumba & Ogunseiju, 2021), earth monitoring (Li *et al.*, 2023), healthcare (Meijer, Uh & El Bouhaddani, 2023), smart buildings (Yang, Lv & Wang, 2022), smart cities (Mylonas *et al.*, 2021), and telecom (Ahmadi, Nag, Khar, Sayrafian & Rahardja, 2021).

In the built environment, various projects have explored the use of DTs for different purposes across different lifecycle phases, including planning and design (Brilakis *et al.*, 2019; Madubuike, Anumba & Khallaf, 2022), construction (Braun, Tuttas, Stilla & Borrmann, 2018; Greif, Stein & Flath, 2020), and the Operation and Maintenance (O&M) (Shi *et al.*; Zhao, Feng, Chen & Garcia de Soto, 2022) phases. This paper specifically focuses on the engineering of DTs in the O&M phase, where they are developed to monitor, enhance, and optimize various aspects of built assets. Given that buildings typically have a lifespan of several decades, DTs must be designed and developed to evolve over time, accommodating changing needs and priorities as circumstances shift.

As discussed in Aissat et al. (2024), DTs in the built environment have primarily been developed in the context of research projects or as prototypes and proof of concepts (PoCs) (Aissat, Beaulieu, Bordeleau, Motamedi & Poirier, 2024b). Their development has typically followed an ad hoc

[1] In this paper, we use the term *system* in a broad sense to refer to an object, process, or system.

[2] While definitions of DT traditionally refer to *Physical Systems*, we prefer the term *Actual System* because DTs can be associated with different types of systems that are not exclusively physical, including Cyber-Physical Systems (CPS), socio-technical systems, and different types of processes.

approach, rather than a systematic software engineering methodology. This lack of structure leads to challenges regarding scalability and evolvability, and more critically, it affects the consistency and predictability of resource planning and outcome management (Shahzad *et al.*, 2022). Our research addresses the need for a systematic approach to facilitate the scalable engineering and evolution of digital twins. In particular, the following three essential characteristics of DTs need to be addressed.

- **DTs are complex software systems.** DTs constitute complex software systems that should be continuously modified and updated to meet evolving user requirements and priorities, while enabling continuous improvement. As such, their engineering needs to be based on a scalable software architecture, and supported by a software engineering process that allows for systematic incremental development/evolution and the automation of process tasks to enable fast and reliable delivery of new DT software versions.
- **DTs are associated with systems composed of multiple aspects.** As discussed in Martens et al. (2024), while a DT is generally defined as a digital representation or replica of an AT, it is more accurate to describe it as a digital representation that encompasses only a subset of the AT's attributes (Mertens *et al.*, Accepted for publication, 2024). For instance, in the context of built assets, a DT may be developed to monitor and enhance various aspects of a building, such as thermal comfort, air quality, energy consumption, security, asset management, and waste management. In this context, the range of aspects addressed by a DT should be incrementally and systematically expanded throughout its lifecycle to incorporate an increasing number of AT aspects (Arup, 2019; Mertens *et al.*, Accepted for publication, 2024; Kamel Boulos & Zhang, 2021).
- **The development of DTs is a multi-disciplinary process.** It is essential to recognize that the development of DTs is a multidisciplinary process that requires collaboration among individuals with diverse expertise, including software and systems engineers, as well as domain experts for the various aspects modeled within the DT. The roles and responsibilities of software and systems engineers primarily involve the software architecture of the DT, the management of heterogeneous data, and the engineering processes necessary to develop,

test, deploy, operate, monitor, and evolve the DT software, along with maintaining the infrastructure on which the DT operates. The required expertise domains vary based on the AT aspects covered by the DT and the set of services it provides. Domain experts are responsible for defining the metrics that need to be monitored and improved, selecting the models and algorithms required to compute these metrics using DT data, analyzing them, and delivering specific *DT Services*.

3.2 DevOps Background

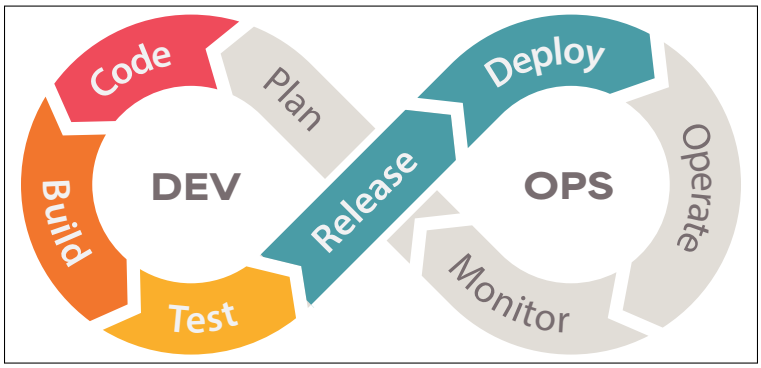


Figure 3.1 DevOps lifecycle loop

Figure 3.1 illustrates the DevOps lifecycle loop diagram as commonly presented in the literature. It is composed of a sequence of phases that include Plan, Code, Build Test, Deploy, Release, Operate, and Monitor. Each phase contributes to the continuous delivery and improvement of software systems by enabling feedback and automation throughout the development lifecycle. In this paper, we focus on a subset of these phases that goes from Code to Deploy, which constitute the core of the software delivery pipeline. Figure 3.2 provides a detailed view of this subset, highlighting the main tasks typically performed during each phase. For instance, the Code phase is described as the sequence of the following tasks : Code, Commit & Push, Static Analysis, Unit Test, Review, Merge to Main. Also, as indicated by the diagram legend, certain tasks, such as Code, Commit & Push, Review, and Merge to Main are carried out manually, i.e., they explicitly require the involvement of human resources, while all of the other tasks aims at being fully

automated to improve efficiency and reliability. Additional details on each of these tasks are provided in 3.5.

Continuous Integration (CI) and Continuous Deployment (CD) play a key role at the core of the DevOps process (Kim *et al.*, 2021). CI aims to integrate each developer's code with that of their peers following a change. The idea is to do this continuously to receive quick feedback, detect integration issues early, such as conflicting or incompatible code, and address them as quickly as possible. This early detection of issues reduces the cost and effort associated with bug fixing later in the development cycle. To achieve this integration, it is essential that the build remains functional after each change. This way, it is possible to identify, for instance, when a change prevents the project from compiling or building. Once a valid build is confirmed, a series of automated verifications can be executed, including static code analysis and unit tests. These quick feedback mechanisms help validate code quality and ensure reliability. Automating these processes improves development efficiency, allowing developers to focus on delivering new features and enhancing functionality, rather than spending time on manual validation steps.

CD builds upon the foundation established by CI. It culminates in the creation of the new release and its automated deployment to production environment. Before achieving this level readiness, the package is automatically deployed to pre-production environment, where it undergoes product integration and end-to-end testing. This stage is crucial for validating the system's behavior under realistic conditions and assessing whether the proposed changes are suitable for release. Through this process, development teams can evaluate the stability, functionality, and overall production readiness of the software. Once validated, a review or approval step, manual or automated, determines whether the changes are promoted to production. By enforcing this practice, CD ensures consistency and reliability across deployments, significantly enhancing the overall quality and robustness of the software product.

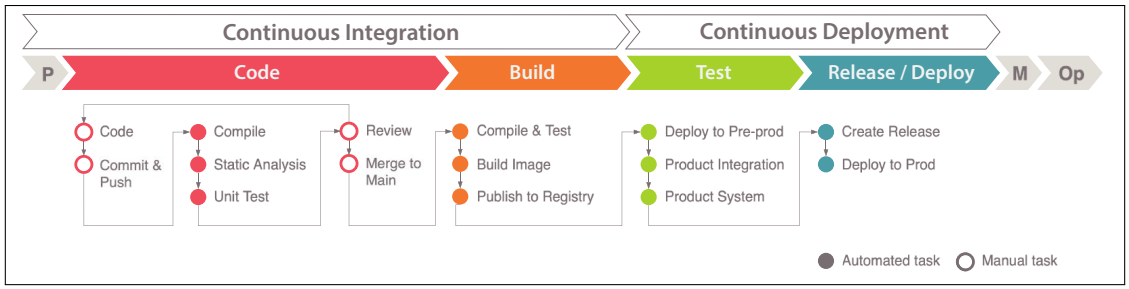


Figure 3.2 DevOps process : from Code to Deploy

3.3 Case Study : GRIDD Lab DT

The development of the JuNo-OPS framework was initiated through a series of projects centered on the creation of DTs for different aspects of the GRIDD laboratory. These initiatives helped uncover key challenges and opportunities associated with developing DTs for built assets, while simultaneously laying the groundwork for the core components of the framework. They also contributed to refining a broader vision for a systematic approach to DT engineering that incorporates software engineering and DevOps best practices while addressing the unique characteristics of DTs. In parallel, several DevOps practices were experimentally integrated into these projects, allowing for early testing and iterative adjustments of the proposed approach. The insights and lessons learned from these projects are discussed in the subsequent sections, as they have directly influenced the future directions and innovations presented, and form the foundation of our proposed framework, which addresses key issues such as modular architectures, efficient data management, and multi-disciplinary collaboration.

From a broad perspective, the work presented in this paper is a stepping stone towards the digitization of a part of ÉTS (École de Technologie Supérieure) University campus in Montreal, Canada. By starting small (i.e. a single DT for a single room), this project is allowing iterative improvements to our research artifacts while grounding this work in practice and contributing to the existing body of knowledge. More specifically, the Design Science Research (DSR), as outlined by Hevner & Chatterjee (2028), is used as research methodology framework (Hevner & Chatterjee, 2010).

3.3.1 GRIDD's Lab description

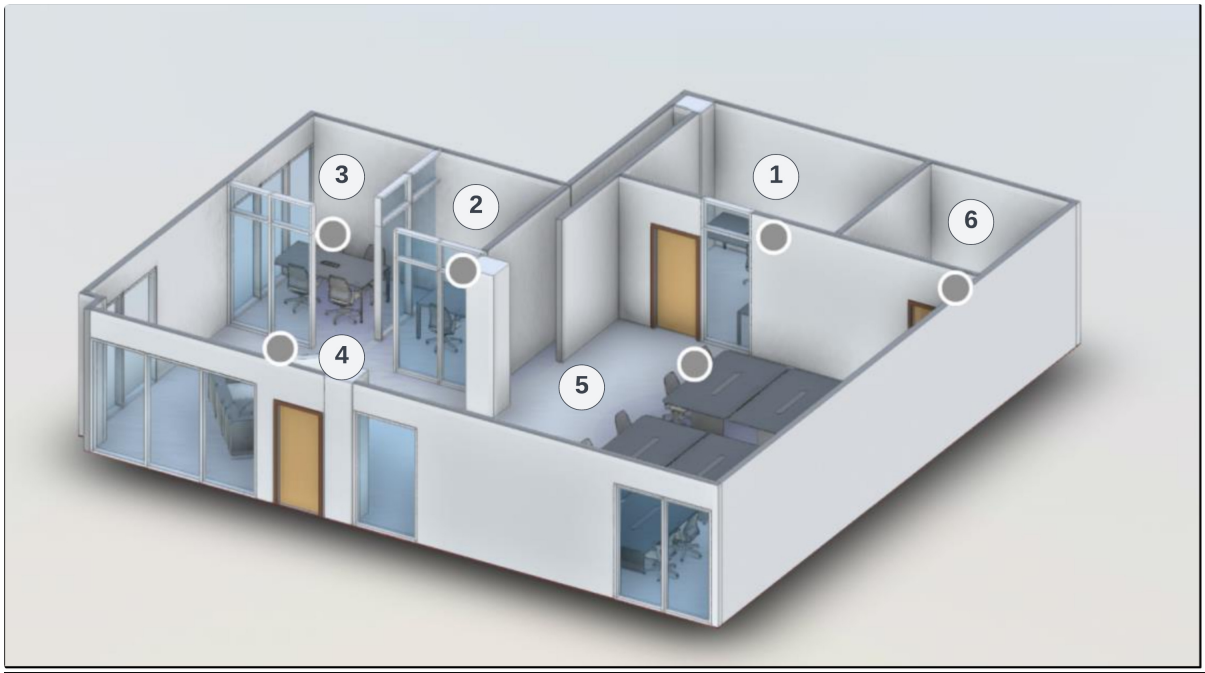
The GRIDD (Groupe de Recherche en Intégration et Développement Durable en environnement bâti [3]) is an ÉTS research group focused on digital transformation, industrialisation, collaboration and sustainability in the built asset industry. The research group is committed to driving sustainable change in the construction industry.

The GRIDD's lab is a multi-functional room at ÉTS that is used for research activities, group meetings, and teaching activities. A 3D model screenshot of the lab is provided in Figure 3.3 (Sharafdin, 2024). The room is divided into six distinct areas, indicated by the numbered dots in the figure. These areas include a closed office, two semi-private office spaces, two open meeting zones, and a storage area. The gray dots in the image represent the locations of various sensors installed throughout the lab for real-time data acquisition, capturing metrics such as temperature, humidity, and occupancy.

Over the last years, a Building Information Model (BIM) of the room has been created and different sensors have been developed and installed in the lab to support different DT research projects. In particular, (Sharafdin, 2024) developed a plug-and-play IoT sensor infrastructure that enables the deployment of different types of sensors. As a result, the lab is currently equipped with a network of sensors that are used to collect data at regular intervals, including temperature, humidity, CO_2 level, and noise level. The lab is also equipped with two cameras : one located at the door to monitor the entries and exits in the lab, and one that is responsible for taking pictures that are used to extract information concerning the equipment present in the lab space (e.g. tables, chairs, computers, monitors, and boards) and the configuration of the space in terms of the location of the tables, chairs, and boards present in the room (privacy requirements are taken into consideration).

The GRIDD's lab has been the source of multiple DT projects that were developed to enhance various aspects of its functionality and management. These projects have contributed to laying

[3] The english translation would be Research Group in Integration and Sustainble Development in the Built Environment



1 Closed office space : A private office for the research assistant responsible for the management of the lab.

2/3 Semi-private office spaces : Two areas designed for collaborative tasks with partial privacy.

4/5 Open meeting spaces : Adjacent zones for informal discussions and teamwork. The larger areas are designed for more extensive gatherings, such as team meetings, presentations, or teaching sessions, where a large screen that also serves as a whiteboard is available. The smaller spaces are suited for intimate meetings or focused discussions with tables for collaboration.

6 Storage room : A dedicated space for storing equipment and materials.

Figure 3.3 3D virtual view and description of the GRIDD's Lab

the groundwork for integrating different components into a cohesive DT system, as well as experimenting with various technologies, such as those used for 3D space representation. These projects have evolved through an incremental, agile-based approach, contributing to the iterative refinement of the DT framework.

3.3.2 Thermal Comfort DT

A Thermal Comfort DT was initially developed. Its purpose is to enhance the environmental conditions by monitoring and controlling factors such as temperature and humidity. Using the existing network of sensors, real-time data are collected, and then analyzed and used to adjust the lab environmental settings to optimize comfort for its users. The DT includes a 3D model representation of the lab using Unreal Engine to visualize the data and the effects of various environmental adjustments.

The architecture of the *GRIDD Thermal Comfort DT* is illustrated in Figure 3.4 together with the *GRIDD Lab (AT)* and its *Environment* using a UML composite structure diagram. The collection of the data needed by the DT to measure and analyze temperature and humidity is made possible by a network of 27 sensors distributed throughout the laboratory. Different components were implemented, to connect these sensors to the cloud-based DT system. The *Sensor Data Hub* component is implemented using a Raspberry Pi™ (RPi) device acting as a Bluetooth Low Energy™ (BLE) hub. The sensors are embedded on Arduino™ microcontrollers based PCB. The access to these devices is illustrated by the *Hum%* and *Temp* ports on the *GRIDD Lab (AT)* component. This configuration ensures a continuous data stream and its storage.

The BIM of the GRIDD Lab, stored in Autodesk Construction Cloud™ (ACC) that is shown as the *BIM Data* component, was imported into Unreal™ Engine using the `Datasmith` plugin to create an interactive *3D Visualization* of the lab. This integration provides a spatially accurate digital representation of the lab, with embedded information about sensor locations and environmental conditions. The exploitation of the BIM laid the foundation for the development of the two services within the *GRIDD Thermal Comfort* DT :

- ***3D Visualization*** : This service allows users to view real-time environmental data within the 3D model of the GRIDD Lab. Sensors are visually represented in the model, displaying current readings for parameters such as temperature and humidity. The visualization allows enhancing understanding by mapping comfort metrics spatially, allowing users to identify areas that requires adjustments. At this stage, such adjustments are not automated : the user

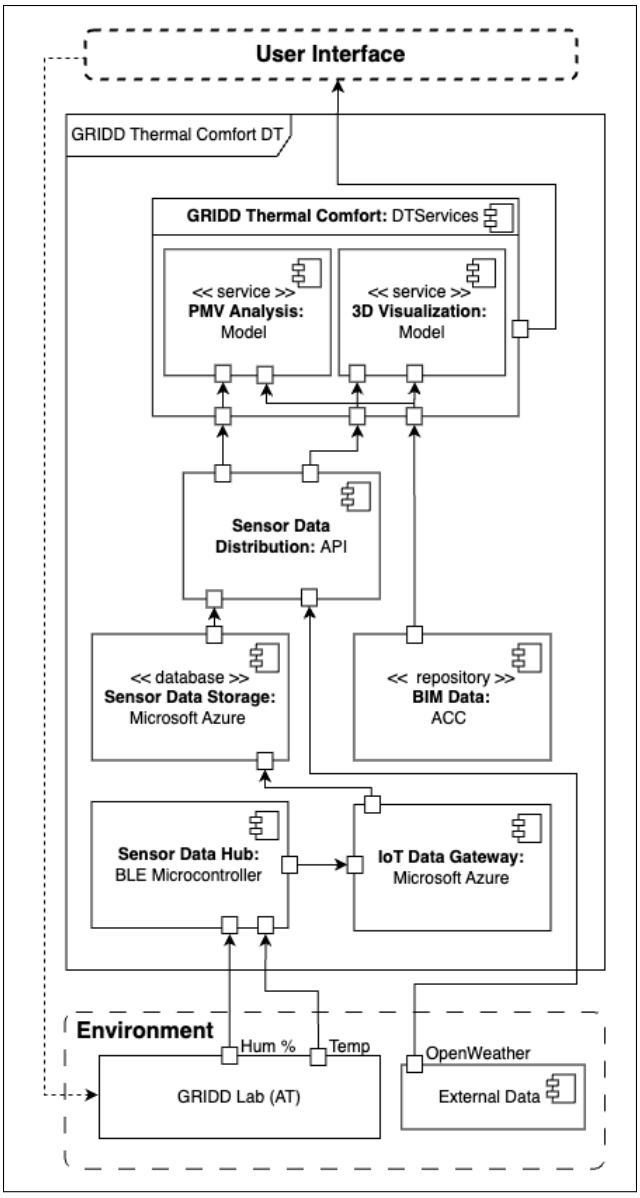


Figure 3.4 Components of the GRIDD Thermal Comfort DT Services for the GRIDD Lab

interprets the visualized data and may manually act on the physical environment (e.g., turning off the heater). This “human-in-the-loop” feedback is illustrated in the architecture diagram by a dotted arrow from the UI back to the GRIDD Lab (AT), distinguishing it from automated communication flows.

- ***PMV Analysis*** : Using the data retrieved through the *Sensor Data Distribution* gateway, this service computes thermal comfort metrics such as Predicted Mean Vote (PMV) and Predicted Percentage of Dissatisfied (PPD) (Fanger, 1970). These calculations, based on ASHRAE standards (of Heating, Refrigerating, Engineers & Institute, 2004), provide actionable insights into the thermal conditions of the lab. The results of the *PMV Analysis* are also integrated into the *3D Visualization*, enabling users to correlate numerical comfort metrics with specific locations within the lab.

Together, these components and services create a comprehensive system for monitoring, analyzing, and visualizing the GRIDD Lab environmental conditions, supporting real-time decision-making and long-term optimization.

3.3.3 Asset Management DT

The Asset Management project focused on real-time inventory monitoring and spatial tracking of objects within the GRIDD lab. Using advanced camera systems and sensors, this project captures and updates information about the types and quantities of objects in the lab (e.g., chairs, tables, computers), ensuring accurate resource management and planning. The data collected is visualized in a 3D virtual environment using Unity™ engine, allowing stakeholders to maintain a precise understanding of inventory levels and object locations, both at the present time and using historical data.

The system relies on a Nvidia Jetson™ device equipped with a custom GPU and running Ubuntu™ Linux. This device, functioning as the *Image Processing* component, as illustrated on Figure 3.5, served to process image data from a connected Intel RealSense™ camera. The camera captured images of the lab space, which were then processed locally using advanced computer vision models. This is illustrated by the *ImageStream* port on the *GRIDD Lab (AT)* component. The Jetson device leverages modules such as `PyTorch` and `Ultralytics` for object detection (e.g., chairs, tables, and white boards) and `PyRealSense2` for depth sensing. This

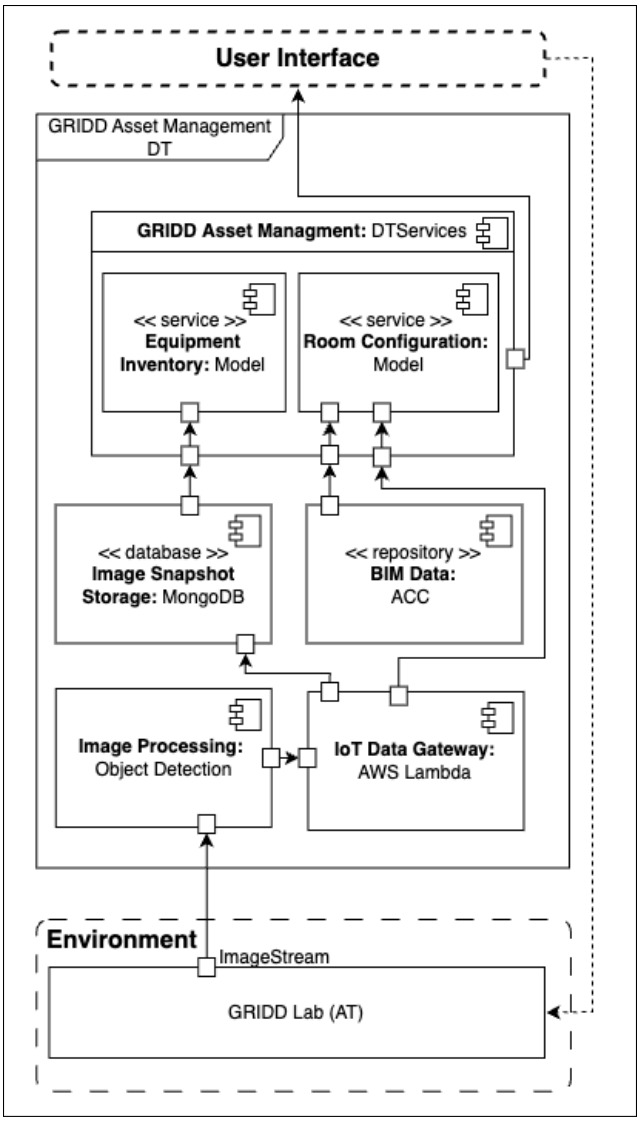


Figure 3.5 Components of the Asset Managmnent DT for the GRIDD Lab

edge processing minimizes bandwidth usage by transmitting only relevant object detection predictions rather than large, raw images to the cloud for further storage and analysis.

Once generated, the predictions are routed through the *IoT Data Gateway* (an Amazon™ API Gateway), which serves as the central hub for data distribution. The *IoT Data Gateway* receives detection results via a `POST` requests and actes as the intermediary between the edge computing device and the downstream services. The incoming data are handled by an AWS Lambda™

function, namely `insertSnapshot`, which processes the metadata and stores it in a NoSQL `MongoDB Atlas` database.

The *Image Snapshot Storage* component, implementing `MongoDB` database, meticulously structures each snapshot to include metadata such as object types (e.g., tables, chairs), spatial positions, and timestamps. This structured storage allows for the tracking of inventory over time, enabling historical analysis and spatial configuration monitoring. The flexible schema of `MongoDB` is particularly beneficial in adapting to the evolving needs of the system, ensuring that new data types or attributes can be integrated without disrupting existing services.

To complement this storage and retrieval mechanism, the *IoT Data Gateway* also facilitates data access for visualization purposes. Unity™, wich is used for the 3D visualization of the *Room Configuration* of the GRIDD's Lab, communicates with the *IoT Data Gateway* through `GET` requests. These requests trigger the `getSnapshot` Lambda function, which fetches the latest inventory data from `MongoDB`. The fetched data are then used to update the 3D model in Unity dynamically, providing an accurate and real-time representation of the lab's inventory and spatial layout.

By integrating edge computing, cloud services, and 3D visualization, the Asset Management DT provided two main services :

- ***Room Configuration*** : This service uses inventory data to track and manage the spatial arrangement of objects within the lab. It provides insights into the current setup and supports planning for future reconfigurations, ensuring efficient use of space.
- ***Equipment Inventory*** : This service monitors and tracks the types and quantities of equipment present in the lab. By maintaining an up-to-date inventory, it supports resource allocation, procurement planning, and loss prevention.

These services offer stakeholders actionable insights and a user-friendly interface to manage and optimize the GRIDD Lab resources effectively. While the system enables real-time monitoring and informed decision-making, the resulting actions, such as rearranging furniture or preparing the space for a meeting, are performed manually by users based on the information displayed.

This human-in-the-loop approach is inherent to the current implementation, where observations made via the UI guide physical interventions within the lab environment. This alignment with cutting-edge technologies allows for robust real-time monitoring and long-term strategic planning.

3.3.4 Integration and evolution of the GRIDD Lab DT

In the next iteration of our research project, the DTs were integrated into a unified DT of the GRIDD Lab, encompassing the two aspects : *GRIDD Thermal Comfort* and *GRIDD Asset Management*. The resulting DT is illustrated at Figure 3.6. Previously developed as independent projects, they now operate together within a cohesive infrastructure. The goal of this integration is to create a robust environment to foster collaboration across teams and to manage the systematic evolution of the system. As in the previous diagrams, a dotted arrow represents the manual user feedback loop between the UI and the physical asset (AT), reflecting the current human-in-the-loop state of the system where control actions remain unautomated.

While the integration process preserved the iterative and agile development approach used in earlier stages, it also laid the foundation for a more cohesive and maintainable DT system. As illustrated in Figure 3.6, the architecture was structured according to the principle of separation of concerns. Organized into two main parts : (1) a collection of independent services that define the logic and functionalities of each aspect, and (2) the *DT Data Management* that handles data acquisition, storage, and distribution. This grouping is only intended to provide a clearer, more coherent view of the architecture that first resulted from the integration.

Within the *DT Data Management*, three logical part were defined to improve architectural clarity :

- **Data Acquisition :** includes the components responsible for collecting raw data from the physical environment (e.g., sensors, cameras).
- **DT Data :** stores structured data used by the DT services (e.g., sensor readings, BIM data).

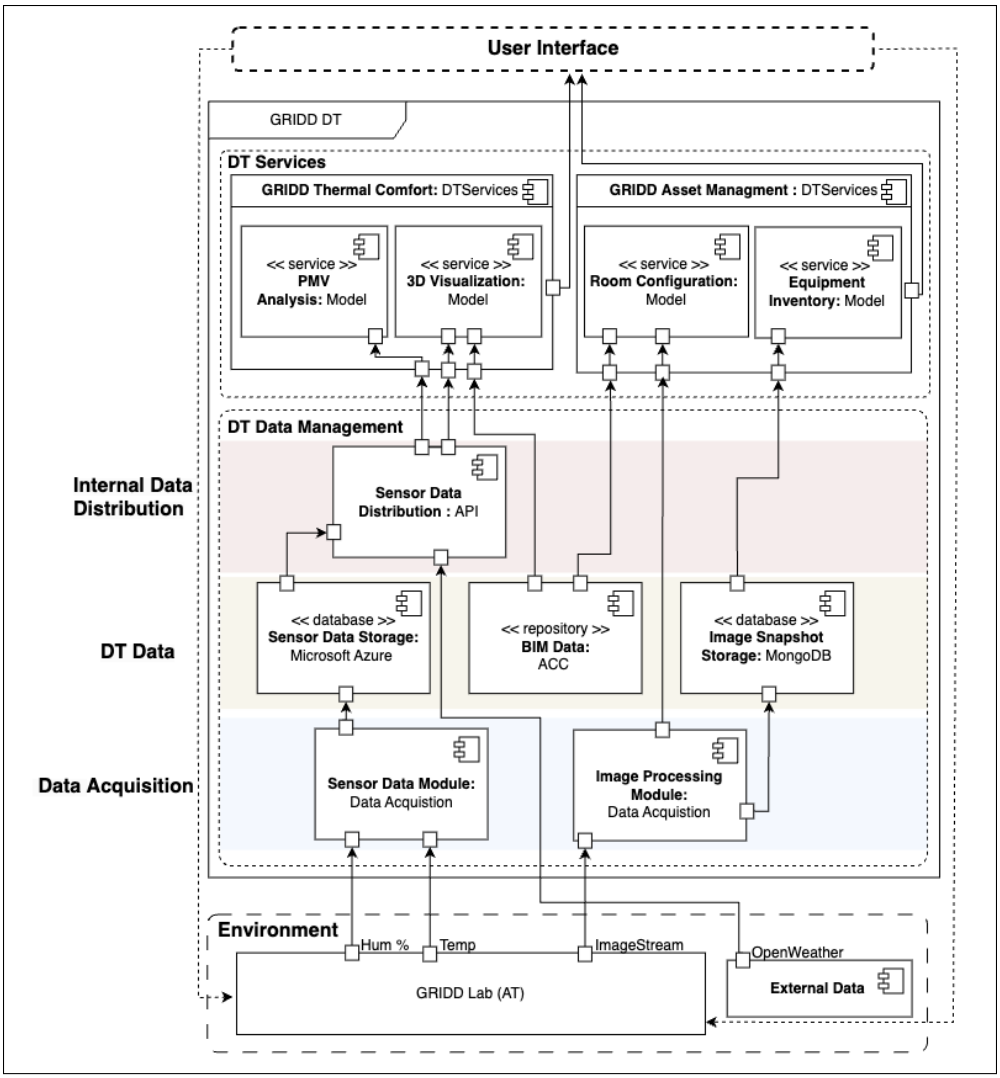


Figure 3.6 Software architecture of the GRIDD Lab DT after integrating the *GRIDD Asset Management* and *GRIDD Thermal Comfort* Aspects

- **Internal Data Distribution :** facilitates communication between data storage and services, primarily through APIs.

To better illustrate how these sub-layers operate in practice, especially in terms of communication and data exchange, we describe the communication workflow implemented for the *GRIDD Asset Management* aspect below.

Communications

To promote a lower coupling, we avoid proprietary or component-specific protocols. Rather, the remote communications between the components takes place over RESTful APIs, using `JSON` messages. In the case of the *GRIDD Asset Management* DT Services, the communications are managed by an AWS API Gateway™ and dispatched to an AWS Function-as-a-Service (FaaS) Lambda Function™ for further processing and storage. In the case of the *GRIDD Thermal Comfort* DT Services, the communications are brokered by an Azure IoT™ gateway. For the front end servicing the UI layer, each micro-service has its own API that handles `GET` requests.

To better illustrate our design choices, we provide additional details on how the communications are handled concretely in the *GRIDD Asset Management* DT Services (Figure 3.7).

Between the AT and the DT :

1. A scheduled "snapshot" is sent from the Jetson™ device, after processing a video frame to detect the relevant objects (the Jetson™ sends a `POST` request)
2. The Jetson's `JSON` data is transformed by the `insertSnapshot` lambda function, who adds an `id` field and a timestamp for indexing in `MongoDB`
3. The document is stored in the Altas™ cloud-based DB service on AWS™

Between the DT micro-services and the UI :

1. The scheduled latest "snapshot" is requested by the Unity™ Engine, through a `GET` request
2. The `getSnapshot` lambda function is activated by the API gateway to return the latest `JSON` document in the DB.
3. Upon reception of the response, the view is updated to refresh the latest positions of the objects in the GRIDD lab

Figure 3.7 *GRIDD Asset Management* DT Communications – Example

As illustrated in figure 3.6, the integration of these, now so-called, aspects has been represented by a first level of abstraction, namely, the use of API gateways to decouple data access from the services. However, this abstraction was limited in scope. For example, while the *Thermal Comfort* aspect leveraged a dedicated data distribution component, the *Asset Management* aspect lacked an equivalent layer, making its integration more tightly coupled with upstream components. This

asymmetry revealed the need for consistent abstraction mechanisms across all DT aspects, not only to streamline development but also to support independent evolution of each aspect.

Moreover, although the architectural separation was beneficial, it also highlighted the importance of anticipating integration requirements from the very beginning. Whether integrating an entirely new aspect or simply adding a service to an existing one, such decisions have a direct impact on data flow, versioning, and deployment processes. These lessons reinforced the need for well-defined integration points and architectural patterns within DT systems.

Ultimately, the consolidation into a unified DT infrastructure served as a valuable learning experience. It allowed us to prototype and validate foundational DevOps practices, including the use of version-controlled repositories and CI/CD pipelines for each service. Overall, the iterative development of the GRIDD Lab DT highlighted the critical need for a scalable, loosely coupled architecture to manage the inherent complexities of DTs. Each iteration underscored the importance of independent, modular components that could seamlessly integrate into a cohesive system. These insights informed the definition of a more mature and scalable architecture presented in the next chapter, where the principles learned from the GRIDD Lab DT are generalized into the JuNo-OPS framework's architecture. This setup not only improved the modularity and maintainability of the system but also positioned the infrastructure to evolve incrementally, whether by enhancing an existing aspect (e.g., extending *Thermal Comfort* with energy efficiency and air quality metrics) or by introducing entirely new ones.

A single DT infrastructure has the potential to better adapt to modifications made to the AT or its environment. These insights informed the definition of a more mature and scalable architecture presented in the next chapter, where the principles learned from the GRIDD Lab DT are generalized into the JuNo-OPS framework.

3.3.5 Other DT-based Projects outside the GRIDD

The groundwork has also been laid in developing other DT-based projects. One point that has been looked at is the visualization of various *DT Services* information into dashboards

(e.g., `Grafana`) to provide a unified view of telemetry data. We have also explored the idea of macro-scale DTs, for instance, we tackled the challenge of modeling a large city-wide fleet of public transit vehicles that are in motion with real-time and frequent updates, and in which we extract high-level aggregated insights and visualizations. This DT is under active development.

3.4 DT Architecture

DevOps promotes the use of microservices to create loosely coupled architectures to help better manage software complexity and enabling the independent development and deployment of software components (Kim *et al.*, 2021). In JuNo-OPS, we adopt a microservices-based architectural approach that leverages DevOps principles to optimize software development and deployment. This approach is designed to manage the inherent complexity of DT software while allowing people working on different DT aspects to operate as independently as possible.

The architecture is divided into tree major layers : *DT Data Management*, *DT Data Communication*, and *DT Services*, all three fueled by data from the underlying AT and its surrounding *Environment*. This includes diverse *External Data* sources.

Figure 3.8 is used as a reference to express the architecture throughout this section. In the sub-sections, details are given on the different layers of the architecture and the design choices made for each. Additionally, it will be used to outline our envisioned future work on these layers, highlighting how they can be further enhanced to maximize their potential benefits in the context of the DTs projects.

3.4.1 The AT (the GRIDD Lab)

The AT and its *Environment* provide the foundational data for the DT. The main data sources include :

1. **Devices and Systems** : The AT integrates a network of IoT sensors and devices that capture real-time data about the *Environment*. In the GRIDD Lab, for instance, various sensors monitor parameters such as temperature, humidity, and CO_2 level. These devices range

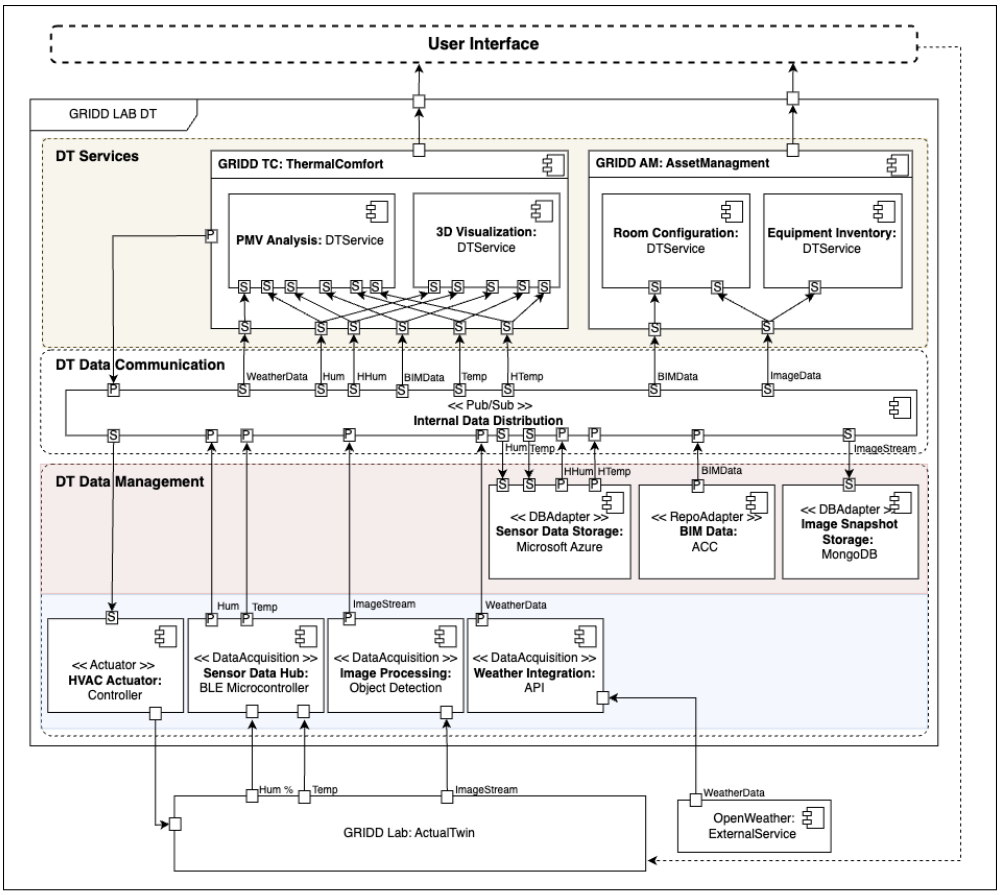


Figure 3.8 Consolidated software architecture for the GRIDD Lab DT

from resource-constrained microcontrollers (e.g., Arduino-based setups) to more capable Single Board Computers (SBCs) like RPi running lightweight containerized environments.

2. **Metadata** : Metadata provides contextual information essential for providing meaning to the data. It can support visualization of information as well as input to physics-based modeling. For example, the BIM of the GRIDD Lab offers a 3D spatial representation of the lab, embedding details about the physical layout along with many other "as built" details like sensors placement.
3. **External Data** : External data sources complement local sensor data to provide a broader context to the DT. These data sources are not fed from the AT itself. In the GRIDD Lab

DT, the OpenWeather API supplies real-time weather information, aiding the system in understanding external environmental influences on indoor conditions.

This multi-source approach to data collection reflects the heterogeneity often observed in smart buildings, as highlighted in the Brains4Buildings framework. (Chamari *et al.*, 2023a)

The GRIDD Lab DT includes one representative of each data category (i.e., devices and systems, external data and metadata). It provides a comprehensive view of the laboratory operations and enable rich data-driven insights, as DT systems aim to do. JuNo-OPS is designed to take into account and effectively integrate these different types of data sources into its architecture. This consideration will enable the framework to support comprehensive and flexible DT implementations in the future.

To facilitate this, the architecture was refactored to follow the principle of separation of concerns, resulting in three core layers :

- **DT Data Management** : which handles data acquisition along with its processing and storage.
- **DT Data Communication** : which facilitates secure and abstracted data exchanges between DT components.
- **DT Services** – where domain-specific microservices responsible for implementing the logic and functionalities of each DT aspect operate.

Each of these layers plays a distinct role while working together using data from the AT and its surrounding *Environment*.

3.4.2 DT Data Management

The role of the *DT Data Management* layer is to collect and store data making them available to the rest of the system, whether it originates from real-time acquisition components or from more stable, pre-existing sources such as the BIM model. It encompasses both data acquisition and storage, handling heterogeneous data types, static or dynamic, transient or persistent, collected

from the AT and its surrounding *Environment*, as well as from *External Data* sources (e.g., Open Weather service).

For instance, transient data acquired in real time from sensors is routed directly to the *DT Data Communication* layer through the `Hum` and `Temp` ports, while persistent data such as historical humidity and temperature readings is retrieved from storage via the `HHum` and `HTemp` ports.

The choice of the specific technologies to use, as well as where the data is stored, depend on the specific DT or combination of DTs that are modeled. In the context of the *GRIDD DT*, the data abstraction is composed of several independent databases and repositories, reflecting the heterogeneity of the data types involved and promoting loose coupling between components.

For instance, `JSON` text documents are stored in a `MongoDB Atlas` instance on AWS™ for the asset detection data (used by the*GRIDD Asset Management DT* services), while Azure TSI™ hosts time series of data for temperature and humidity. The 3D model *BIM Data* is retrieved from Autodesk Construction Cloud™, a COTS service.

As illustrated in Figure 3.8, the architecture introduces dedicated *data access adapters* for each storage source, rather than depicting the storage technologies themselves. These adapters encapsulate the specifics of how data is retrieved, enabling a clean separation of concerns. This design choice enhances modularity and simplifies future evolution : if a storage solution needs to be replaced (e.g., migrating from Azure TSI to another time-series database or switching BIM providers), only the corresponding adapter must be updated, leaving the rest of the architecture unaffected.

3.4.3 DT Data Communication

The role of the *DT Data Communication* layer is to provide a unified and efficient entry point for secure data sharing between DT components that allows abstracting the complexities of accessing individual data sources. By centralizing data access, this layer ensures that the *DT Services* can seamlessly retrieve the necessary data for their operations. It provides a unified

abstraction for information sharing that shields the upper layer from the complexities of data acquisition, processing, and storage.

The *DT Data Communication* layer is implemented by an *Internal Data Distribution* component based on the *publish/subscribe (pub/sub)* asynchronous communication paradigm, which allows producers (e.g., data acquisition components) to publish data to logical *topics*, and consumers (e.g., service components) to subscribe only to the topics relevant to their domain. This design reinforces the principle of *separation of concerns* by allowing each microservice to access only the data it needs, without having to manage where or how it is generated or stored.

To ensure efficient filtering and routing of messages, the topic structure follows a hierarchical naming scheme (e.g., `sensors/temperature/room1`), enabling fine-grained subscriptions. In addition, a form of *content-based filtering* is envisioned to further allow services to receive only the messages that meet specific criteria (e.g., temperature threshold exceeded), which improves system scalability and responsiveness.

The introduction of this intermediary communication layer supports scalability, resilience, and maintainability across the entire DT system, while laying the groundwork for more advanced data orchestration patterns in the future.

3.4.4 DT Services

The role of the *DT Services* layer is to group all the services provided by the DT in a single layer and decouple them from the technical issues related to data management.

The *DT Services* layer builds on the foundation provided by both the *DT Data Management* and *DT Data Communication* layers, offering domain-specific functionalities that leverage the collected and processed data. Each aspect of the DT corresponds to a broader functional domain and is composed of multiple services. These services operates as independent microservices, adhering to a modular architecture that ensures loose coupling and high cohesion.

These services are containerized using `Docker`, which ensures that each microservice is self-contained, including all necessary dependencies for deployment, and can run consistently across different environments, whether on the cloud or at the edge. This modularity allows services to be updated, scaled, or replaced independently, providing significant flexibility and ensuring that the system remains agile in the face of evolving requirements.

`Docker` containers are used to run software components on the cloud and at the edge whenever possible. This forms an easily deployable build artifact which can also be checked in GitLab repo for version control. By allowing independent deployability of smaller yet complex functional components, we tackle the DT's complexity and multiplicity of aspects more easily.

The decision to execute some of the services locally on edge devices stems from practical constraints like bandwidth limitations and cost considerations. For example, instead of transmitting raw images to the cloud, a Nvidia Jetson device can locally process camera feeds for object detection, transmitting only actionable data to the downstream systems in the cloud. This design not only reduces network load but also supports privacy by limiting the sharing of sensitive raw data.

The microservices-based architecture naturally aligns with Conway's Law by reflecting the division of responsibilities among teams (Conway, 1968). Each team is tasked with managing a specific aspect of the DT, fostering strong internal cohesion while minimizing dependencies between services. This separation of concerns promotes efficient collaboration and supports automation, a key principle of the DevOps approach.

Generally speaking, each microservice is independently developed, deployed and executed (e.g, in isolation). Each microservice models a part of a DT (e.g., in Figure 3.8, the *Equipment Inventory* and *Room Configuration* services in the *GRIDD Asset Management* DT Services), and is developed by different teams modeled around "business domains" (e.g., building occupant comfort, computer vision, BIM management). This is a conscious choice as it takes advantage of the natural tendency for each team to develop its part of a solution within a larger organization, as described by the Conway Law. This keeps internal cohesion to a maximum while, ideally,

keeping external coupling to a minimum. Those are highly desirable traits to maintain said independent deployability and support automation, which are both in turn, corner stones in a DevOps approach.

Moreover , this layer is designed to ensure the independence and isolation of domain expertise. Standards, best practices, and specific requirements of each domain are contained within this layer, ensuring they are unaffected by the underlying architectural decisions for deployment. Domain-specific considerations are integrated directly into the modeling and implementation of services, which remain the responsibility of domain experts. Meanwhile, the deployment of architecture built on containerized microservices using `Docker` is automatically managed, enabling independent development and deployment. This separation ensures seamless alignment between domain expertise and technical implementation while preserving the flexibility and scalability of the overall system.

This architecture also anticipates the future need for interactions between different DT aspects. APIs acting as facades can be implemented to enable seamless communication and integration between aspects without requiring domain experts to navigate the complexities of another domain. At present, such interactions have not been necessary within the GRIDD Lab DT, as the aspects and their respective services are orthogonal. Each aspect operates independently, addressing its specific objectives without overlapping functionality or requiring cross-aspect collaboration.

3.4.5 User Interface (UI)

The role of the UI layer is to provide a unified and coherent point of access to the various aspects of the DT. It acts as the primary interface between end users and the DT system, enabling the visualization, interaction, and exploration of data produced by backend services. The APIs of individual services, potentially complemented by aspect-level facade APIs, serve to integrate and deliver data seamlessly to the UI. This ensures that information originating from various DT

aspects is unified and presented in a coherent manner, facilitating user interaction with the DT system.

At the moment, each aspect has its own dedicated UI. The UI stacks chosen by each team vary in implementation details, but generally revolves around dashboards such as Grafana and PowerBI™, complemented with 3D model visualisations that spatially contextualize information using the Unity Engine or the Unreal ™. These visualizations enhance user understanding and interaction with the system. To maintain consistency and efficiency, limiting the variety of UI stacks supported by the framework is considered beneficial. This approach encourages the reuse of code and services across different aspects of the DT while still allowing customization within each domain.

To improve coherence and reduce redundancy, a unified UI may be envisioned in future iterations of the framework. This common interface would centralize access to all DT aspects, offering a streamlined experience to end users. To support this evolution, a dedicated *UI Adapter* component could be introduced. Its role would be to handle the communication between the unified UI and the various backend services, abstracting the underlying architecture and protocols. Communication could be handled through WebSocket connections for real-time updates, or RESTful APIs for more traditional request-response interactions, depending on the nature of the service.

It is also important to distinguish between 3D visualizations used as part of the data processing pipeline and those designed for displaying results to end-users within the user interface. For instance, some visualizations may serve as intermediate tools for spatial analysis or simulation during the execution of a service, while others are intended to present final final outputs displayed as part of the service offering within the user interface. While a unified UI will centralize interactions, multiple visualization tools may coexist, depending on the functional needs of each DT aspect.

Overall, the DT architecture serves the purpose of the DT while supporting its evolution. It facilitates the implementation of DevOps practices to effectively manage the complexity of DTs. Being loosely coupled, with well defined team boundaries and strong internal cohesion, the framework addresses the multidisciplinary nature of DT development while enhancing scalability, deployability, and automation.

3.5 DevOps Infrastructure

One of the goal of the DevOps infrastructure is to automate as many phases of the overall software engineering process as possible. By streamlining repetitive tasks and improving the efficiency of the overall software process using a variety of technologies and software tools, the framework enhances productivity, consistency and quality, providing all the typical benefits of DevOps, including fast feedback and continuous learning.

This supports the seamless integration and management of heterogeneous components, such as IoT devices and external sources protocols or cloud services, which are essential for the dynamic and evolving nature of a DT. The automation provided by the framework benefits everyone involved in the DT engineering process, including software engineers who are developing code for specific DT platform components and domain experts, with no software engineering background, who can contribute their expertise to the core functionality of the DT without being constrained by software engineering details.

In this section, we describe three main elements of the DevOps framework : (1) the structuring of the `GitLab` infrastructure, including groups and repositories, to support the development of microservices-based (multi-aspect) DTs, (2) the continuous integration (CI) pipelines pipelines and (3) the continuous deployment (CD) pipelines. The current version is implemented in GitLab and will be released in open source (under an MIT license) before the end of 2025.

3.5.1 GitLab Infrastructure Organization

To meet the specific needs of the microservice-based architecture as presented, a multi-repo structure was chosen in the GitLab code repository. This approach leverages the loosely coupled nature of microservices by storing each microservice in an independent repository with its own versioning. This avoids imposing artificial interdependencies that could complicate the deployment and maintenance of individual components from each functional system within the *DT Services* of the digital twin.

In a microservices architecture, each component is designed to function as autonomously as possible, with its own development and deployment cycle. By structuring the GitLab repository around this independence, we can isolate each microservice and thus maximize the autonomy of each team working on a specific service of a functional system. This allows domain experts to contribute their expertise in a targeted area without worrying about environments outside their field of interest. This decoupling also enables simpler and more efficient CI/CD pipelines, without needing to orchestrate the deployment of services unaffected by a specific change or update.

3.5.1.1 GitLab Groups and Project

To further enhance organization and manageability, GitLab groups are leveraged to logically cluster repositories based on their function and role within the DT system. This hierarchical structure ensures clear boundaries between different system components (microservices) and layers, and facilitates efficient management of the entire codebase.

Taking the example of the *GRIDD DT* as shown in Figure 3.9, each microservice related to the *GRIDD Thermal Comfort* aspect, such as the *PMV Analysis* and the *3D Visualization*, is managed within its own repository. Similarly, the microservices for the *GRIDD Asset Management* aspect, including the *Equipment Inventory* and *Room Configuration* Models, each reside in their respective repositories. These repositories are then grouped under GitLab Groups representing the larger functional systems, such as *GRIDD Thermal Comfort* and *GRIDD Asset Management*

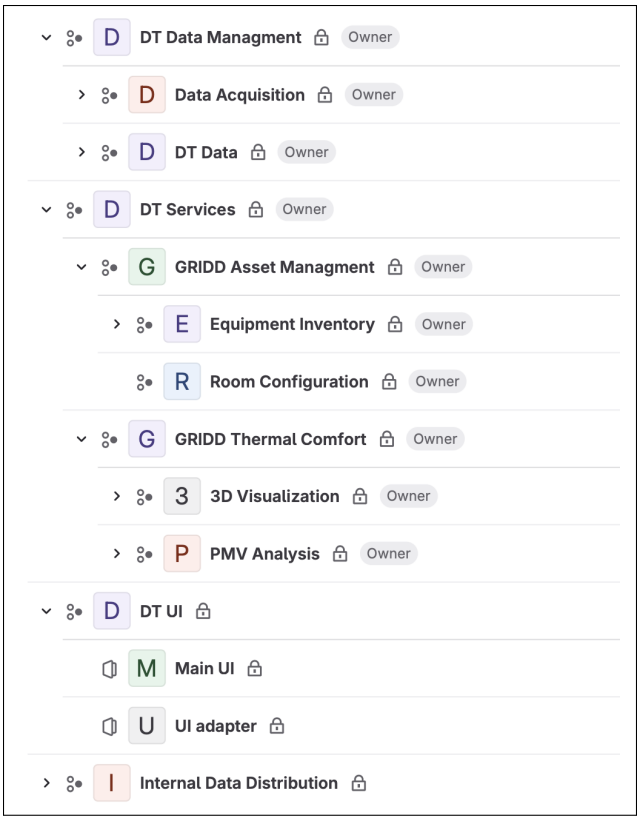


Figure 3.9 GitLab Group structure for the GRIDD DT system

aspects, providing logical organization and access control. This multi-repo organization not only enables more granular version management, but also targeted, agile deployment according to the specific evolutions of each service.

Additionally, repositories supporting the *DT Data Management* layer, including those responsible for *Data Acquisition* (e.g., Sensor Data Gateway and Image Processing), data storage (e.g., Sensor Data, MongoDB Snapshot, and BIM Repository), and internal data distribution (e.g., the Pub/Sub system) are organized separately. These repositories remain distinct from the *DT Services* repositories, where domain experts contribute, ensuring a clean separation of concerns.

This structural distinction also extends to deployment environments, such as `Kubernetes` clusters. Here, container orchestration is configured to provide separate access and deployment control for the *DT Data Management* layer and *DT Services*, ensuring that data handling

components remain isolated from service-specific functionalities. This separation enhances both security and scalability by limiting access to containers based on the roles and responsibilities of the teams involved.

Furthermore, the organization of repositories and deployment environments is based on the modular DT architecture defined for the GRIDD DT system. Layers such as the UI and the DT Data layer are designed to be accessed through adapters, promoting future flexibility. For example, the user interface, which, in this case, was implemented as a React web application, can be updated, replaced, or extended without requiring changes to the underlying DT Services components even the DT Data Management components. This design choice enhances the maintainability, scalability, and long-term evolution of the JuNo-OPS framework.

3.5.2 Git Workflows for Effective Digital Twin Development

Various workflows are used to support DT development across the different levels of the software architecture. A `Git` workflow is a process structured by a branching model that defines how code changes are introduced, reviewed, and integrated into a `Git` repository, along with the various steps required to deploy the application to different environments production. These branching models manage feature development, testing, and deployment in an organized and collaborative manner at both the *DT Services* level and *DT Data Management* level, as illustrated by the GitLab group structure in Figure 3.9.

At the *DT Services* level, GitFlow is adopted to manage deployment pipelines for various microservices while ensuring seamless integration. This branching structure is tailored to the iterative nature of the services layer, enabling robust testing and validation before updates reach production. The branching model includes :

- **Feature branches** : Created from the development branch, these branches are used to develop new features or implement specific changes. Once the feature is complete and tested, it is merged back into the development branch, ensuring isolated development without impacting the stability of the main codebase.

- **Development branch** : Serves as the integration base for all contributions from feature branches. It contains the latest stable code that has passed initial testing and is ready for further validation in staging environments.
- **Release branches** : These branches are created from the development branch when a new release is prepared. Final testing is conducted in a production-like environment, and the bug fixes or urgent necessary tasks are performed (with no new feature development allowed). Once all of the bugs identified during the final test cycles are resolved, the release branch is merged into both the main and development branches to ensure consistency.
- **Main branch** : Contains stable, production-ready code. Only thoroughly tested and validated changes from the release branch are merged here. A merge into the main branch triggers the deployment pipeline to the production environment.
- **Hotfix branches** : Created from the main branch for urgent fixes to address critical issues in production. Once resolved, the fixes are merged back into both the main and development branches to maintain alignment between production and ongoing development.

At the *DT Data Management* level, a GitLab Flow is employed to streamline the development and integration of foundational components. The workflow simplifies the processes for adding or modifying elements such as data gateways, storage systems, and *Internal Data Distribution* mechanisms, making it easier to reach a stable state for these critical components. The branching structure includes :

- **Main branch** : This branch contains production-ready stable code. Only validated changes ready for production deployment are merged here. A merge into the main branch triggers the deployment pipeline to the production environment.
- **Development branch** : The development branch serves as the integration base for all contributions from developers. It contains the latest stable code that has passed initial testing and is ready for staging. Changes from feature branches are merged into the development branch, where they are further tested and validated before being promoted to the main branch.
- **Feature branches** : Local branches, originating from the development branch, are created and used by developers to develop new features or modify existing ones. These branches

allow for isolated development or bug fixes before the changes are merged back into the main branch. For example, developers might create a feature branch to enhance a specific functionality or resolve an issue, without impacting the development branch until their work is ready for staging tests.

This setup provides a simplified workflow for the *DT Data Management* level, allowing the team to focus on finalizing stable and scalable solutions for data acquisition, storage, and distribution. Future works will explore refinements to this layer to ensure its long-term compatibility and robustness.

By adopting these complementary workflows, the Digital Twin system balances the dynamic evolution of the *DT Services* with the foundational stability of the *DT Data Management* level. This structured approach supports both innovation and reliability, ensuring the Digital Twin's readiness for future enhancements and scalability.

3.5.2.1 GitLab Merge Request

The Merge Request (MR) is the mechanism used in GitLab to ensure that any code modification is peer-reviewed before the code is accepted to be merged and integrated into the collaborative codebase. In a DevOps process, the creation of a merge request is automatically triggered whenever a developer submits a code modification that has successfully passed through the development pipeline, which will be discussed below.

Utilizing GitLab MRs within a branching model-based structure ensures controlled development and maintains code quality, keeping the main branch always ready for production. At the microservice level, developers work on their local feature branches, and with push permissions on the main branch set to "no one", once ready, they are forced to submit their changes through the merge request process. This approach guarantees that only verified and validated changes reach the release branch for further testing in a pre-production environment before being merged into the main branch, facilitating a smooth deployment to production. Therefore, to support continuous deployment and delivery, it is crucial to keep the Main branch always functional.

Finally, by setting up a GitLab structure and using workflows specific to each repository, the management of the various CI/CD pipelines becomes clearer and easier to implant at the different levels up to the complete DT.

3.5.3 CI Pipelines

Continuous Integration (CI) is composed of two main pipelines at the microservice level. A Dev pipeline, that is executed in the Code phase, followed by a Build pipeline.

The goal of the Dev pipeline is to ensure the quality of the code before it is reviewed (in the Review task). It is triggered each time a developer makes a commit or a push and runs on the developer's local branch at the microservice level. It automates the execution of the Compile, Static Analysis, and Unit Test tasks, each with the following purposes :

- **Compile** : This task performs the compilation of the code to ensure there are no syntax errors or other issues, and produce the executable.
- **Static Analysis** : This task uses static code analysis tools to check the quality of the code and its adherence to coding standards and conventions.
- **Unit Test** : This task automates the execution of the set of unit tests associated with the code that has been modified. It aims at verifying that individual components function as expected.

The goal of the Build pipeline is to package the microservice and publish its artifact in a binary [4] registry. It is triggered by the merge of the submitted code into the Main branch following its acceptance by the reviewer(s) during the peer-review process (Review task). The sequence of tasks executed in the Build pipeline is as follows :

- **Compile & Test** : The submitted code is first integrated with the rest of the codebase on the Main branch to ensure there are no integration conflicts. Although the code has already been compiled and tested on the developer's local feature branch in the Dev pipeline, it must be recompiled and retested (Unit Test task) after being merged into the Main branch to ensure that no problems have been introduced by the integration. Static code analysis (Static

[4] The term binary refers to compiled or executable versions of software, such as executable files, libraries, or compiled code, that are necessary for running applications

Analysis task) on the other hand doesn't have to be re-executed since the code itself has not changed.

- **Build Image** : In this task, the `Docker` image associated with the modified microservice is created. For this purpose, a base image must first be selected (from an existing library of `Docker` images), considering factors such as performance, footprint, security, and the source/provider of the image. The `Docker` file is then written using the selected base image, to which the microservice's source code and dependencies are added to finally build the image by running the `docker build` command.
- **Publish to Registry** : The new `Docker` image is then uploaded and stored in a binary registry, e.g. DockerHub [5]™ or Artifactory [6]™.

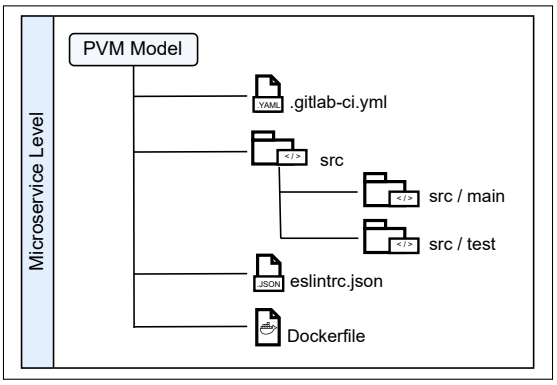


Figure 3.10 PMV Model microservice repository structure

Figure 3.10 uses the PVM Model microservice of the *GRIDD Thermal Comfort* functional system as an example to illustrate the structure and content of the GitLab repository that is used to support execution of the two CI pipelines (i.e. Dev and Build pipelines). It contains the following elements :

- A **GitLab CI configuration file** called `.gitlab-ci.yml` that specifies the various stages of the CI pipeline at the root of the repository. This file is located at the root of the repository

[5] https ://hub.docker.com/

[6] Artifactory is a DevOps solution for managing and distributing software binaries and artifacts

and contains different triggers to distinguish between the development pipeline and the microservice build pipeline and is located.

- The **source code directory** (`src`) containing two subdirectories : `src/main`, which holds the code of the microservice, and `src/test`, which includes the set of unit tests for the microservice that are executed as part of the pipeline.
- A **static code analysis configuration file**, in this case the `.eslintrc.json` file that is used by the ESLint [7] static code analysis tool. This file provides the set of rules to be applied to the repository's specific language as part of the Static Analysis task.
- A **`Dockerfile`**, that contains a series of instructions, such as specifying the base image, copying files, and running commands, to automate the process of building the `Docker` image for the microservice. This file is located at the root of the repository.

In addition to the elements shown, the repository structure is also prepared to support communication mechanisms toward the DT Data Communication layer. This includes, for example, serverless functions such as Lambda functions, responsible for operations like publishing messages to specific topics managed by the Pub/Sub broker. SThese mechanisms are managed within the same monorepository alongside the APIs and domain models constituting the logic of a service associated with a specific aspect of the digital twin. When serverless deployments are required, the Serverless Framework [8] is used, with a dedicated `serverless.yml` configuration file placed at the root of the repository. This configuration is triggered together with the main CI/CD pipeline to automate the deployment of both the containerized microservices and the serverless communication mechanisms.

The pipeline thus ends by pushing the `Docker` image in a registry, which will act as our artifactory where we store our versioned artifacts.The next stage, Continuous Deployment (CD), focuses on automatically deploying these artifacts into the appropriate environments following their validation and publication.

---

[7] https ://eslint.org/

[8] The Serverless Framework is an open-source deployment framework that simplifies the building and deployment of serverless applications across various cloud providers. See : https ://www.serverless.com/

3.5.4 CD Pipelines

The CD pipeline aims to automate the deployment of software applications and microservices in the different pre-production and production environments, where they can be executed, tested, and released. CD pipeline automates the deployment of all microservices within the DT system across pre-production and production environments. These pipelines streamline the process of delivering software artifacts, testing them in controlled environments, and releasing them for production use. By employing such pipelines, the deployment process becomes efficient, scalable, and consistent, supporting the iterative development and evolution of the DT system.

3.5.4.1 Deployment Workflow

The CD pipeline employs two distinct branching strategies, tailored to the specific needs of the *DT Services* microservices and the *DT Data Management* microservices. These branching models ensure that each layer of the system benefits from a workflow optimized for its unique requirements.

In the staging deployment phase, code from the develop branch is deployed to a staging environment. This environment is designed to replicate the production setup as closely as possible, providing a realistic setting for comprehensive system-level tests. These tests include end-to-end testing, integration testing, user acceptance testing, and many more. The staging environment serves as a final checkpoint where any identified issues can be addressed, ensuring that the new updates are stable and integrate seamlessly with the existing functionalities. For *DT Services* microservices, this phase is particularly crucial as it validates the coherence of new features with the broader system.

When it is time to deliver the artifact to production, a specific artifact version is selected for release. This selection triggers the same CD process, but against the live environment, ensuring that the deployment is tested and validated under real-world conditions. For production deployment, only code from the main branch is deployed to the production environment. The transition to this branch is tightly controlled, with merges allowed only after all tests in the

staging environment are successfully completed. Once deployed, the production environment reflects the latest stable version of the system, ensuring a reliable and high-quality experience. To maintain traceability, `Git` tags are used to mark specific production-ready versions within the repository.

In microservices managed under the GitFlow strategy, release branches play a key role in the deployment process. These branches are created from the develop branch to allow for final adjustments, such as resolving minor bugs or making last-minute changes, without affecting ongoing development. Once these changes are validated, the release branch is merged into both the main and develop branches to ensure consistency and alignment across all environments.

The deployment workflow is automated to streamline the transition between environments. Merging code into the develop branch automatically triggers the pipeline to deploy the updates to the staging environment, where system-level tests are executed to verify the changes. Upon successful testing, the release branch (or directly the main branch in the case of GitLab Flow) is promoted to production. This automation ensures that the deployment process is efficient, controlled, and consistent, reducing the risk of disruptions and maintaining the quality of the *DT Services* and *DT Data Management* microservices.

To facilitate continuous deployment in containerized environments, tools such as `Kubernetes` [9] and `Helm` [10] can be used to customize scalability. `Kubernetes` orchestrates containerized applications across a cluster of machines, ensuring efficient resource allocation based on application needs. `Helm` charts further simplify the management of deployments, providing templates for `Kubernetes` resources. This includes a new suite of tests that have yet to be fully implemented, further enhancing the robustness and reliability of the deployment process.

[9] https ://kubernetes.io/

[10] https ://helm.sh/

3.5.4.2 Cloud-based Microservice Deployment Pipelines

Regarding Microservice deployment pipelines, the goal is to develop a library of pipelines that supports all of the different nodes/targets that are required/used in the pre-production or production environment. This process begins by pulling the artifact from the Artifactory™ and delivering it to one or more environments as necessary.

For *DT Services* microservices, deployments are typically cloud-based, leveraging platforms such as AWS™ or Azure™. These environments provide the scalability and flexibility needed for dynamic updates and the integration of new features. Each microservice aims to be independently deployed to minimize interdependencies, ensuring that updates to one service do not disrupt others. At this point, we have developed pipelines for AWS™ and Azure Cloud™ environments.

3.5.4.3 Edge-based Microservice Deployment Pipelines

As previously mentioned, for latency-sensitive use cases, such as real-time processing or services requiring immediate responsiveness, edge deployments will eventually need to be considered. Deploying microservices closer to the source of data generation can reduce latency, improve system efficiency, and enhance overall user experience (Satyanarayanan, 2017). For instance, the "*Image Processing*" service of the acquisition layer (Figure 3.6) is deployed on an edge device, such as a Nvidia Jetson™ or Raspberry Pi (RPi). These deployments are optimized for latency-sensitive tasks, reducing the need to send raw data to the cloud. Edge deployment enables the processing real-time camera feeds *directly at the source*, and only the extracted features need to be sent upstream, thereby minimizing bandwidth usage and improving system responsiveness. We have currently developed pipelines for RPi, Jetson™, and Arduino™™ IoT devices.

Building automated deployment pipelines for edge devices is a distinct use case with specific challenges. In implementing such pipelines, one challenge was met in the form of strict network administration rules. Direct access to a connected device through a standard SSH session (using agentless Ansible for automation) was not allowed by the firewall and local area

network parameters. Using a VNC connection mediated through a cloud-based VNC server was considered to circumvent the problem, but it is not an "automatable" solution. Therefore, GitLab Runner installed on the edge device itself solved the issue because it acts as an agent to establish communication from within with GitLab server. This allowed the set up of a fully automated CD pipeline using GitLab CI and `Docker`. Figure 3.11 describes the sequence of actions that were executed for the Nvidia Jetson running Jetson Linux based Ubuntu 20.04. The RPi devices steps are similar as they use Linux based Ubuntu 20.04 as well.

Edge device initial set up :

1. `Git` clone the Object Detection Model (ODM) repository
2. Install `Docker.io`
3. Create a `Docker` file with `3.12-bookworm` Python image
   - Include the ODM's Python modules dependencies (i.e. `NumPy`, `PyTorch`, `Ultralytics`, `PyRealSense2`, etc.) as requirements.
   - Run `Python3 -m main.py` script
   - Push the `Docker` image to DockerHub™
4. Install a GitLab Runner instance following GitLab's guidelines
5. Register the Runner on the device and on GitLab's platform

Writing `.gitlab-ci.yml` file to define pipeline configurations :

1. Tags : "Jetson" to help identify this part of the pipeline
2. Job1 : Pull `Git` repository head
3. Job2 : Build the `Docker` image and push it to DockerHub™
4. Deploy stage : Pull the `Docker` image from DockerHub™ and run it.

Figure 3.11 Sequence of actions for deploying on the NVIDIA Jetson

While this approach is acceptable on Linux-based devices, it is not ideal for scaling as having one agent for each edge device makes the infrastructure heavier to maintain. Potential solutions to this challenge are discussed in Section 3.7.1. Further, as the DT system evolves, we will explore hybrid deployments, balancing cloud and edge-based solutions.

3.6 Related work

The concept of Digital Twins (DTs) has evolved significantly, particularly in the context of built assets, encompassing buildings and infrastructure. This section reviews DT frameworks in smart

buildings and infrastructure, DT architectures and the integration of DevOps practices for the engineering of DTs.

DTs in built assets

The application of DTs in the built domain is relatively nascent but rapidly growing. Early implementations in Smart campuses focus on integrating BIM with real-time data from IoT sensors to create comprehensive digital representations of buildings. The Cambridge campus case study, UK,(Lu *et al.*, 2020) demonstrated how DTs could be leveraged to improve asset maintenance and FDD, highlighting the potential for DTs to enhance the operational efficiency of buildings through real-time monitoring and predictive analytics. Shi et al. (2015) looked into applying the idea of “Continuous Commissioning” through FDD and energy efficiency in a similar fashion on Carlton campus, Canada (Shi *et al.*). In their SLR, Hosamo et al. (2022) stress the necessity of a "comprehensive method of implementing BIM and IoT with data processing to build a Digital Twin" (Hosamo *et al.*). On Aalto University campus, Finally, Khaajavi et al. (2019) explored a detailed implementation of sensor node mesh on a facade (Khajavi *et al.*, 2019) while Dave et al. (2018) focused on IoT deployment of the DT with a user-centric perspective (Dave, Buda, Nurminen & Främling), allowing contributions from the community. The growing body of research in this area focuses on specific aspects of DT, i.e. on the model delivering and optimizing the services themselves. The research presented in this paper focuses on supporting model delivery, their evolution and the people who develop them through a microservice as part of a DT infrastructure.

Djebali et al. (2024), DTops as a DT design framework for Smart Grid DTs based on the DevOps approach and related methodologies (Djebali, Guerard & Taleb). This paper surveys the opportunities of DT applications to Smart Grid combined with a vision toward a DT design process for Smart Grids that could be generalized. They support the premises that the application of DevOps in DT is highly relevant and compatible.

In the related body of research from the built asset community, data lakes or data warehouses architectures styles appear to be the dominant in DT architectures. They usually imply data pipelines are put in place to collect AT data and serve a single aspect thereof. Those are similar to the first implementations of DTs made in the GRIDD Lab as described in the Case Study section. However, the next iteration of architecture shown in Figure 3.6 proposed looser coupling that microservices architecture and DevOps processes call for.

DT Architectures

Lu et al. (2020) propose a multi-tier architecture that separates the concerns similarly to the OSI model (Lu *et al.*, 2020). While the “Service Layer” looks like the *DT Services* described in our research, it seems to offer little separation of concerns with the "Data/Model Integration Layer". The architecture proposed by Chamari et al. (2023) is clear in this regard while it also focuses greatly on solving the data integration and interoperability challenge through technologies such as protocol adapters and ontologies (Chamari, Pauwels, Petrova, Dubbeldam & Jong). However, issues such as data synchronization and quality remain significant hurdles. Those issues are to be addressed as part of our future work as well.

Marosi et al. (2022) propose a data analytic platform to serve real-time Business Intelligence and batch Exploratory Data Analysis using Azure IoT™ "hot-warm-cold" path logic for data flow (Marosi *et al.*, b). The path selection is based on the timeliness of the data. Their reference architecture includes "Custom Components" support which could be used, in some ways, in place of one DT/microservice presented in our research. Unfortunately, the components are not fully described. As depicted, they seem to be in the lower levels of the architecture, allowing for flexibility, but leading us to conclude that they are not considered as a first-class concept. Fundamentally, the platform is a data analytic platform first, to which complementary components can be added for extended DT capabilities. We argue that data analytics of an aspect of the AT tied to a purpose should be treated as a microservice, thus allowing for clear separation of concerns, and simplifying the deployment and management of the infrastructure in production.

In another distinct paper Marosi et al. (2022) applies Reference Architecture (RA) (AKA template/cookie cutter) in the context of DT (Marosi *et al.*, a). They deployed an end-to-end Machine Learning RA in test infrastructure as a case study. This paper complements the work being done in our research in many ways as it deals with DT architecture and deployment, but the focus is on the reusability quality of the RA. While we expect this quality to become highly relevant in the future, it is out of scope in our research.

3.6.1 DevOps practices in engineering of DTs

DevOps practices in engineering of DTs

The integration of DevOps practices in the development and operation of DTs is a relatively new area of research. DevOps, which emphasizes Continuous Integration and Continuous Deployment (CI/CD), automated testing, and monitoring, can significantly enhance the lifecycle management of DTs.

López-Peña et al. (2020) illustrate how DevOps can be adapted for IoT systems, providing fast and continuous feedback on system availability and performance (López-Peña *et al.*, 2020). Applying these principles to DTs can improve their responsiveness and adaptability, ensuring that they remain an accurate reflections of their physical counterparts (i.e. the AT).

A growing body of work, mainly coming from the Model-Based System Engineering (MBSE) community, tries to transfer pure software-based DevOps practices to CPS. Those show the ability and potential benefits of applying selected DevOps capabilities in safety-critical embedded systems (Dobaj *et al.*, 2022). They tend to use DT in CPS to enable DevOps (Ugarte Querejeta, Etxeberria & Sagardui; Mertens & Denil, 2021) as opposed to DevOps for the DT as a system. The concerns revolve around accelerating deployments and run-time telemetry to gain real time data for future design improvements. (Hugues *et al.*). Combemale & Wimmer (2020) follows the same path, but add a testing phase to the exploration (Combemale & Wimmer). Within the large scale AIDOaRt project, Eramo et al. (2021) bring AI tools into the mix (Eramo *et al.*).

Tisi et al (2021) bring the ideas of a MBSE based System of Twins (SoT) to support and manage a DT through the whole life cycle (Tisi, Bruneliere, de Lara, Di Ruscio & Kolovos, 2021). The aspects described in this article are specific DTs for each "constituent" of the system described by the authors. The composition of all DTs results in a SoT. It is suggested that the use of a "meta-environment for building specific DTs" to support domain experts, but the prime function of this platform is to support the modeling of the whole system. Also is stated that, having many domain experts building a SoT independently is "unrealistic if we consider the growing organizational and technical complexity of CPSoS" (Cyber Physical System of Systems). They should instead, be subject of a controlled and orchestrated hierarchy of the meta-model that support the meta-environment. As made apparent in our research paper, we agree with the creation of a platform to support domain experts, but DevOps has shown that more control over individual teams is not the best way to manage the growing complexity of systems and organizations. However, this controlling meta-model approach may be necessary for dependability in safety critical CPS.

Kostromin & Feoktistov (2021) introduced a framework towards automation of infrastructure provisioning and environment configuration using Multi-Agent System in the context of digital twins for built assets (Kostromin & Feoktistov). In that sense, they cover a very specific and limited scope of the DevOps approach.

In summary, as far as we understand the current body of knowledge about DevOps practices in engineering of DTs, no research address DevOps approach as a mean to enable collaboration

3.7 Discussion

This section discusses some of the main challenges that we encountered in the design of our DevOps DT framework.

3.7.1 Deployment on Distributed Heterogeneous Edge-Cloud Platforms

As seen in Section 5.3, while most components of the DT models that we developed were deployed on the cloud, some components were deployed at the edge. This deployment heterogeneity raised several challenges.

Communication Network

A major challenge that had to be solved is related to the fact that the edge-based Jetson device was located behind a highly restricted network, complicating the automated deployment of updated software components. In our case, the problem was temporarily solved, but the solution may not apply everywhere. Besides, pulling the code and building the `Docker` image on the same runtime node may seem like unnecessary extra work, but to make sense of this approach, we need to consider the following. Once the network access problem is permanently solved, the GitLab Runner will likely run on a local VM or public Cloud. Just like the approach described in this paper, it will generate an artefact (`Docker` image), store it under version control to be tested and should deploy to many identical edge devices instances across the ÉTS campus/network. Any SBC running a Linux OS acting as edge device is expected to apply a similar approach to CD pipeline automation.

Handling communication losses is also a challenge. How to update the microservice onto an edge-based device in the event of a network failure, or in the event of a partial OTA DFU which could lead to device bricking ? In the same manner, how to handle a failing device, or how to handle fluctuating networking conditions (e.g., WiFi or cellular).

Power Management

In an effort to lower the total power consumption of the devices, we may consider slim down distributions of Linux such as Ubuntu™ Core or Yocto. The pipeline for updating OSes, especially those implying a kernel recompilation, has yet to be defined and may prove to be a challenge.

µC are more energy efficient, but more specialised firmware are sensitive to communication losses during update process.

We came to realize that the deployment process in itself needs to consider distributed systems-related challenges. On slim down versions of Linux on edge devices may be more energy efficient, but `Docker` containers use may become a challenge deploy and run on such OS distribution.

3.7.2 Domain Experts with Non-Software Engineering Backgrounds

As previously mentioned, the engineering of DT associated with a complex system requires the collaboration of a group of people with different background, work culture, and expertise. At a minimum, the teams multi-disciplinary teams developing DT need to get basic knowledge of Git and testing.

Git

Some degree of training on `Git` is necessary on the tool itself, but also on best practices of how to use it. Indeed, in the DevOps approach, the use of `Git` in a specific way (e.g. trunk based with short lived feature branches) can be challenging at times for developers themselves. This could lead to misunderstandings for non-developers and significantly impact the whole process.

Testing

Writing proper tests to validate the quality of one's code is a specialty. Some cultures favors that developers write their own tests rather then handing off the task to quality control team, but this already implies that the developer are software engineers. Furthermore, test suite must be maintain and renewed. This challenge may only be answered by a software engineer being embedded in each micro-service team.

3.7.3 Data Security and Privacy

An architecture that comprises several components at various layers (e.g., edge and cloud) must consider data security and privacy aspects. At the basic level, traditional secure communications mechanisms should be put in place, such as SSL and authentication – for instance, in our *GRIDD DT*, all communications use HTTPS. In addition, due to privacy implications, it might be advisable to process some sensitive data locally to reduce the exposure risk. In the case of the *GRIDD Asset Management* DTs, given that we run the image recognition algorithm at the edge, the raw images containing sensitive data are not transmitted over the network. Further, protecting, and ideally shielding the data sources (e.g., databases) is also important in our *GRIDD DT*, all databases (DBs) are accessed only through API endpoints (i.e., the external components cannot directly access the DBs). Finally, ensuring that the microservice endpoints themselves are secure is paramount ; otherwise, this could lead to disastrous consequences, such as an attacker accessing private data of the DT, and even performing unauthorized/dangerous actions on the AT itself.

3.7.4 Integration of DTs

The case study presented in this paper illustrated the integration of two orthogonal DTs, *GRIDD Thermal Comfort* and *GRIDD Asset Management*. In this instance, the integration is straightforward because the two DTs do not interact with each other. However, in general cases, the DTs that require integration are often not orthogonal, necessitating a more careful approach to their integration. For example, aspects like thermal comfort, air quality, and energy consumption typically have conflicting goals that need to be systematically resolved. In such case, approaches like the one presented by Martens et al. (2024) can be used to conduct a systematic integration of the individual DTs (Mertens *et al.*, Accepted for publication, 2024).

3.8 Conclusion

In this paper, we presented JuNo-OPS a DevOps framework for the engineering of DTs for built assets. The framework has been developed, tested and validated in the the context of a multi-function room at ÉTS. It directly addresses the three essential characteristics described in the introduction (DTs are complex software systems ; DTs are associated with systems composed of multiple aspects ; and the development of DTs is a multi-disciplinary process). The main focus of the paper is on the microservices architecture of the Digital Twin (DT) software and the DevOps infrastructure that is used to support the development, continuous integration, and continuous delivery of the DT software. By automating main phases of the DevOps process, the JuNo-OPS allows domain experts with no software engineering background to benefit from the best practices of DevOps and software engineering and concentrate on the core value that they can contribute to the engineering of DTs. This way fruitful collaborations can be established in such a way that a set of complementary expertise and domain knowledge can contribute to the development of DTs that address a broad range of system aspects.

Future Work

While the JuNo-OPS has been developed and partly validated within the context of the GRIDD Lab DT, the next validation phase will extend to upcoming research projects, including the development of a DT for a research climatic room and a DT for an experimental floor architecture project, as part of ÉTS's broader strategy to digitize its campus. These projects have already been launched and will progress over the coming months, contributing to the incremental evolution of the framework. We are also leveraging collaborations with other research institutes and universities to further enhance the adaptability and scalability of the framework through various use cases and perspectives.

Furthermore, building on the foundation laid by JuNo-OPS, several areas of future research and development are envisioned, including :

- **Develop a generalized DT Architecture Framework** : One of our main objectives is to generalize the JuNo-OPS architecture developed for built assets, as described in this paper,

into a DT architecture framework that can be applied across various types of systems and application domains.

- **Develop a customizable DT Data Management layer** : The development of DTs requires addressing a range of issues related to the *DT Data Management* layer discussed in sections 3.3 and 3.4 : data acquisition, data storage, and internal data distribution. Our main objective is to develop a library of DT Data Management components/services that can be used to develop DTs in different application domains, including : (1) a set of IoT device configurations for data collection in various physical environments ; (2) a set of data services to ensure data confidentiality, integrity, and availability ; and (3) a a well-defined and stable publish/subscribe mechanism mechanism with a generalizable topic structure for internal data distribution, enabling efficient real-time updates for *DT Services*.
- **Release JuNo-OPS as an Open-Source Framework** : To enable the adoption of JuNo-OPS across multiple application domains and foster community-driven evolution, we plan to release it as an open source framework in the upcoming months.

Finally, future research should continue to explore the synergies between DTs and DevOps, focusing on overcoming the challenges of data (integration, synchronization, quality management, security and privacy) and people (domain experts and software engineers) to realize the full potential of DTs in the built environment.

Declarations

Funding

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada.

CONCLUSION ET RECOMMANDATIONS

Le développement des jumeaux numériques appliqués aux environnements bâtis demeure confronté à un manque notable de structuration méthodologique, entravant leur évolution systématique, leur reproductibilité et leur intégration dans les pratiques du secteur. Cette recherche a été initiée en réponse à ce constat, avec pour objectif de proposer une approche DevOps adaptée à ce contexte, fondée sur trois axes complémentaires :

- **Une méthodologie logicielle** pour encadrer le développement systématique et itératif des jumeaux numériques, favorisant une meilleure planification, traçabilité et maintenabilité des solutions mises en œuvre.
- **Une architecture microservices** favorisant la modularité et la maintenabilité par un découplage fonctionnel des composantes du système et l'indépendance des développements pour une meilleure collaboration interdisciplinaire à grande échelle.
- **Une infrastructure automatisée**, soutenant les processus clés afin d'assurer l'intégration et le déploiement continus (CI/CD) et garantir la cohérence des pratiques tout au long du cycle de vie logiciel.

Afin d'identifier précisément les problématiques et les besoins associés au développement des jumeaux numériques, une phase d'analyse exploratoire a été menée au sein du laboratoire GRIDD de l'ÉTS. Celle-ci s'est appuyée sur l'observation directe de projets en cours, des entretiens avec les différents profils impliqués et l'analyse critique de leur fonctionnement.

Il en a émergé le *framework* JuNo-OPS, conçu pour répondre aux besoins spécifiques identifiés et structurer le développement des jumeaux numériques en environnement bâti. Ce *framework* permet d'orchestrer les contributions interdisciplinaires autour d'une architecture logicielle modulaire, en séparant clairement le développement des services liés aux différents aspects fonctionnels du jumeau numérique de la gestion des données, assurée par une couche de

communication centralisée reposant sur un modèle *publish/subscribe*. Cette organisation favorise une répartition claire des responsabilités, permettant aux experts métier de se concentrer sur le développement des fonctionnalités spécifiques, tandis que l'infrastructure prend en charge les aspects logiciels transversaux. Parmi ceux-ci figurent l'organisation des dépôts et des groupes GitLab, structurés pour gérer les accès et les responsabilités selon le principe de séparation des préoccupations, ainsi que la mise en place de pipelines d'intégration et de déploiement continus (CI/CD) visant à automatiser divers processus de développement. L'ensemble contribue à réduire la complexité logicielle à laquelle sont exposés les experts non techniques, tout en assurant un accès unifié aux données et une cohérence globale des déploiements.

JuNo-OPS est actuellement utilisé dans le cadre de nouveaux développements visant à enrichir les fonctionnalités du jumeau numérique du GRIDD, servant à la fois de banc d'essai et de démonstrateur pour l'étude de ses apports et de ses possibilités d'amélioration continue.

Orientations futures

À travers cette étude, plusieurs axes d'amélioration et d'approfondissement ont été mis en lumière, suggérant des perspectives pour la poursuite des travaux :

Le cadre conceptuel même des jumeaux numériques, incluant leur modélisation, pourrait bénéficier des travaux en cours sur l'enrichissement de notations telles que la notation DarTwin (Mertens *et al.*, Accepted for publication, 2024), qui vise à mieux définir et représenter les aspects fonctionnels et structurels des JN, en permettant de raisonner sur le système, ses objectifs, ses propriétés et son mode de mise en œuvre. L'architecture modulaire sur laquelle se base l'infrastructure encadrant le *framework* JuNo-OPS mérite également d'être explorée davantage afin de mieux répondre aux exigences en matière de sécurité, de performances et de gestion des flux de données et d'évolutivité, notamment en lien avec les objectifs d'évolutivité, en particulier pour des systèmes interconnectés à grande échelle. Par ailleurs, des travaux sur l'automatisation

complète du déploiement du *framework* JuNo-OPS sont envisagés, afin de faciliter son adoption grâce à une infrastructure capable de se configurer automatiquement à partir de paramètres clés fournis. L'intégration de mécanismes de surveillance continue des performances des jumeaux numériques pourrait également venir alimenter les phases de planification des itérations futures, en cohérence avec la boucle de rétroaction propre aux approches DevOps.

Enfin, le *framework* JuNo-OPS, développé dans le cadre de cette recherche, présente un potentiel pour structurer de manière systématique le développement des jumeaux numériques, bien que son déploiement à plus grande échelle et ses bénéfices restent encore à démontrer. Les expérimentations se poursuivent dans le cadre d'infrastructures réelles telles qu'une chambre climatique instrumentée ou le nouveau bâtiment F de l'ÉTS, afin de valider les principes établis et d'éprouver la flexibilité du *framework* dans des cas d'usage variés. Par ailleurs, des collaborations sont en cours avec plusieurs établissements à l'international, notamment l'University of Antwerp, l'Eindhoven University of Technology (TU/e) et Østfold University College, dans une démarche d'échange et de co-développement visant à enrichir mutuellement les approches, méthodes et outils liés aux jumeaux numériques, dont l'extension du *framework*. À terme, la mise à disposition en open source du *framework* et de ses composants vise à encourager la collaboration avec la communauté scientifique et industrielle, et à soutenir un enrichissement progressif des méthodes et outils proposés.

Ainsi, cette recherche marque le point de départ de l'élaboration d'une démarche DevOps adaptée aux spécificités des jumeaux numériques en environnement bâti. Elle ouvre la voie à de nouvelles avancées vers des jumeaux numériques plus accessibles, robustes, modulaires et évolutifs, positionnés comme des leviers essentiels pour la gestion intelligente et durable des infrastructures de demain.

ANNEXE I

DÉTAILS DES ENTRETIENS RÉALISÉS

Le présent guide d'entretien a servi de base pour orienter les discussions menées dans le cadre de la phase exploratoire de cette recherche. Il regroupe les grandes thématiques abordées ainsi que les principales questions posées aux participants lors des entretiens semi-structurés.

Tableau-A I-1 Guide d'entretien utilisé lors des entrevues semi-structurées

Thématique abordée	**Questions principales**
Contexte du projet	Sur quel projet travaillez-vous actuellement, et pouvez-vous en décrire brièvement le contexte ? Comment ce projet a-t-il été lancé ? Quels étaient ses objectifs initiaux, ses phases principales et les parties prenantes impliquées ?
Rôle et responsabilités	Quel est votre rôle spécifique dans ce projet ? Quelles sont les tâches que vous avez réalisées ou réalisez concrètement ?
Tâches et quotidien de travail	Quelles sont les activités que vous effectuez au quotidien ? Avec quels outils ou environnements travaillez-vous ?
Défis et problèmes rencontrés	Quels problèmes techniques, organisationnels ou humains avez-vous rencontrés dans vos tâches ? Quelles en ont été les conséquences ou limitations sur le projet ?
Alignement avec les objectifs	En quoi ces défis ont-ils influencé l'atteinte des objectifs initiaux du projet ? Certaines ambitions ont-elles été révisées ou abandonnées ?
Besoins et pistes d'amélioration	Que faudrait-il mettre en place, selon vous, pour faciliter votre travail ou améliorer le processus de développement du jumeau numérique ? Quelles ressources, outils ou approches manquent actuellement ?

Le tableau suivant présente une synthèse des échanges réalisés dans le cadre de la phase exploratoire de cette recherche. Il regroupe les profils des participants, la nature de leur implication dans les projets liés aux jumeaux numériques, leur champ d'intervention, ainsi que les méthodes utilisées pour recueillir leurs perspectives.

Tableau-A I-2 Aperçu des échanges réalisés dans le cadre de la phase exploratoire

ID	**Profil**	**Nature de l'implication**	**Champ d'intervention (rôle dans les projets)**	**Méthode de recueil**
E1	Étudiante à la maîtrise	Projet de recherche sur le développement d'un système de questionnement basé sur les données d'un actif d'infrastructure	Exploration de méthodes d'interrogation et d'exploitation intelligente des données issues des JN pour la gestion	Entrevue semi-structurée
E2	Étudiante à la maîtrise	Projet de recherche sur la mise en place d'un système de gestion documentaire intégré aux jumeaux numériques	Organisation, structuration et modélisation des documents techniques et administratifs liés aux composants d'actifs dans un JN	Entrevue semi-structurée
E3	Étudiant en fin de bac (génie logiciel)	PFE sur le jumeau numérique du GRIDD	Développement back-end et intégration des données via des API	Entrevue semi-structurée + observation
E4	Étudiant en fin de bac (génie logiciel)	FE sur le jumeau numérique du GRIDD	Modélisation 3D et intégration visuelle du jumeau numérique	Entrevue semi-structurée + observation
E6	Ressources encadrante (professeurs, superviseurs, collaborateurs)	Encadrement ou appui ponctuel aux projets liés aux jumeaux numériques (accompagnement technique et stratégique)	Intégration de l'IoT, modélisation de données, gestion du cycle de vie de l'information, BIM, CPS, transformation numérique du secteur bâti	Observations + retour d'expérience + suivi de projet

ANNEXE II

PUBLICATIONS SCIENTIFIQUES

A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins

Sara Aissat, sara.aissat.1@ens.etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Jonathan Beaulieu, jonathan.beaulieu.2@ens.etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Francis Bordeleau, francis.bordeleau@etsmtl.ca
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), Canada

Ali Motamedi, ali.motamedi@etsmtl.ca
Department of Construction Engineering, École de Technologie Supérieure (ÉTS), Canada

Érik Poirier, erik.poirier@etsmtl.ca
Department of Construction Engineering, École de Technologie Supérieure (ÉTS), Canada

Abstract
Digital twins (DT) have emerged over the last decade as a potential solution to monitor and improve different aspects of built assets. However, little emphasis has been placed so far on the iterative development and evolution of DTs. In software engineering, DevOps has established itself over the last decade as the leading paradigm to improve software processes. This paper proposes a DevOps approach for the development and evolution of DTs for built assets. It focuses on three main aspects: (1) Software methodology to support the systematic and iterative development and evolution of DTs; (2) Microservice architecture to enable the independent development and composition of different aspects of DTs; and (3) DevOps infrastructure to support the development, integration, and deployment of DTs. The articulation of the proposed approach is based on experience gained in the development of various DTs in the context of built assets and other application domains.

Keywords: DevOps, digital twin (DT), built assets, BIM, microservices, software methodology

1 Introduction

Digital twins (DT) have emerged over the last decade as a potential solution to enable the monitoring and improvement of several aspects of built assets across the different stages of their lifecycle (Jones et al., 2020). In this paper, we focus on the Operation and Maintenance (O&M) phase of build assets. The services offered by existing DTs in the built environment include the monitoring and improvement of occupant comfort, energy consumption (Halhoul Merabet et al 2021), air quality (Chenari et al 2016), and security (Camposano et al 2021), among many others. Their development has required addressing significant challenges related to different aspects of the digitalization of built assets, including integration of heterogenous data from different sources and device types, data security and secure access to guard against unauthorized access and cyber-attacks, as well as deployment of software components on heterogeneous execution platform/environments composed of IoT devices and cloud. However, little emphasis has been placed so far on how to concurrently develop and maintain DTs while increasing their range of functionalities in an iterative and evolutionary manner to adapt to ever changing environments, and evolving user requirements and objectives.

DevOps has established itself over the last decade as the leading paradigm to improve software processes in software engineering (Forsgren et al 2018). This approach has allowed

software organizations to increase their agility to adapt to environments that are constantly evolving in order to deliver solutions faster, with higher quality, and that are adaptable to user needs. DevOps aims to integrate the development (Dev) and operations (Ops) phases into a seamless, end-to-end, continuous process flow and emphasizes continuous integration and delivery, and quick feedback loops. It enables continuous improvement through automation and monitoring at every stage, from design to deployment, including planning, development, testing, integration, release preparation (building/packaging), and monitoring. Its adoption by industry leaders such as Amazon, Netflix, Google, and Facebook has led to spectacular progress (Kim et al 2021).

So far, DTs in the built environment have been mainly developed in the context of research projects or for prototypes/Proof of Concepts (PoC). Their development has been typically done in an ad hoc manner, following a known and recognized solutions architecture yet not in a structured nor through a defined and validated process. This leads to issues with scalability and replicability of development approaches and more importantly consistency and predictability of resource planning and management of outcomes (Shahzad et al 2022). The problem addressed by our research is the lack of systematic approach to support the scalable engineering and evolution of DTs.

In this paper, we propose an approach based on the best practices of DevOps to support the systematic development, deployment, and operation of industrial DTs in the context of built assets. Three key areas are presented and discussed in the paper: (1) Software methodology to support the systematic and iterative development and evolution of DTs; (2) Microservice architecture to enable the independent development and composition of different aspects of DTs; and (3) DevOps infrastructure to support the development, integration, and deployment of DTs. Regarding the DevOps infrastructure, we focused on three main aspects: the role of Git as a version control system to provide a single source of truth and support team collaboration, Continuous Integration (CI) to support the development phases, and Continuous Deployment CD to support the release and deployment phases. The proposed approach is based on experience gained in the development of various DTs in the context of built assets and other application domains.

2 Background

2.1 Digital Twin of the GRIDD Lab

A DT is a digital representation of an actual system, also called the Actual Twin (AT), that is dynamically and constantly updated with system data, and provides a set of services related to the system. Examples of DT services include the monitoring of specific properties, detection of issues, simulation, analysis of what-if scenarios, and data analytics to identify correlations between different system properties. One of the main advantages of using a DT is that it can provide advanced services that could not be provided without the use of a digital representation, e.g. simulation, advanced analytics, and AI/ML.

The GRIDD[1], a research laboratory at ETS, focuses on advancing digital transformation, industrialization, collaboration, and sustainability. Committed to driving sustainable change, particularly in the construction industry, the GRIDD research group has increasingly focused on innovative projects that explore the potential of DTs. The GRIDD Lab is the multi-functional room at ETS that is used for both research and teaching activities. It has been the source of multiple DT projects that aimed at enhancing various aspects of its functionality and management. These projects have allowed experimenting with various technologies and contributed to the definition of requirements and challenges for the development of a systematic approach and framework to support the engineering of DTs.

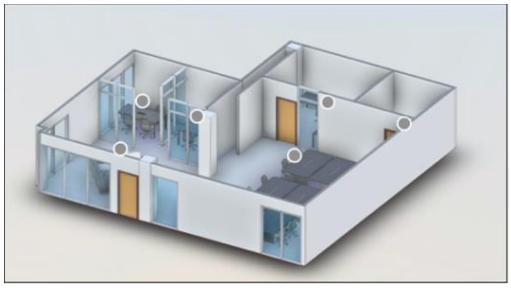
To illustrate the approach presented in this paper, we use the example of a DT that has been developed for the purpose of improving the comfort of the ETS GRIDD Lab. Figure 1 shows three

[1] Groupe de Recherche en Intégration et Développement Durable en environnement bâti. The english translation would be Research Group in Integration and Sustainable Development in the Built Environment.

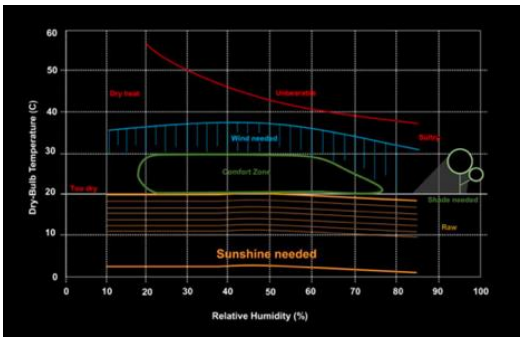
aspects of this DT: a) a 3D virtual view [2] of the GRIDD Lab build based on Building Information Model (BIM) data; b) a PMV (Predicted Mean Vote) (Fanger 1977) thermal comfort chart; and c) a diagram outlining the high-level software architecture of the DT in relation to the GRIDD Lab (AT) and the User Interface that provides access to DT information (or insights).

In the current implementation, the GRIDD Lab is equipped with a network of sensors that collects data related to different aspects of the lab (CO^2, noise level, temperature, and humidity) at regular intervals and stores them in the Sensor Data timeseries database of the DT[3]. Also, a BIM model of the lab is available.

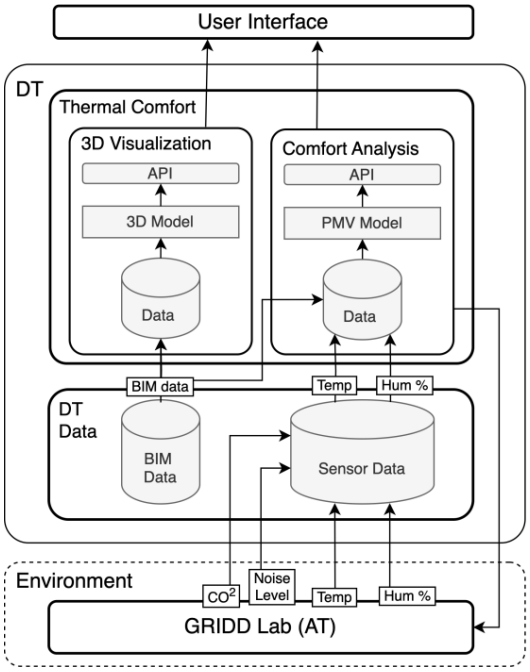
From a software architecture perspective, the DT is composed of a DT Data component and a set of components responsible for delivering the DT services. The DT Data component, which can also be viewed as a data lake, is responsible for storing the AT data and making them available to the different service components. In the example of Figure 1.c, it contains the BIM data of the GRIDD Lab and the data provided by the sensors (Sensor Data) deployed in the lab. From a service perspective, the DT provides two main services related to the improvement of comfort: a 3D Visualization of the lab and its properties related to Thermal Comfort; and a Comfort Analysis service. Each service component is composed of a database, one or more models used to produce the required service based on the available data, and an API that provides external access to the services. The data used by each service is a subset of the data available from the DT Data component, e.g. while the DT Data component contains raw sensor data regarding CO^2, noise level, temperature, and humidity, the Comfort Analysis service only uses temperature, and humidity. Similarly, the BIM data used by each of the service are subsets of the complete BIM Data.



a) 3D virtual view of the GRIDD Lab



b) PMV thermal comfort chart



c) DT software architecture

Figure 1. DT to improve occupant comfort of the GRIDD Lab

2.2 Challenges in the development of DT for built assets

The development of digital twins for built assets involves several layers, each presenting a variety of challenges that must be systematically addressed to ensure successful systems that are not only scalable but also able to keep pace with changing environments and user requirements,

[2] The 3D virtual model was implemented using the Autodesk Revit tool and Autodesk Construction Cloud.
[3] In our current implementation of the DT, this database is deployed in Azure cloud.

while retaining their reliability and efficiency. In this paper, we focus on the software engineering aspects of the development of DTs for build assets . These challenges are indicative of the inherent complexities associated with DTs, highlighting the need for expertise beyond traditional construction knowledge, in favor of a multidisciplinary approach to software engineering. These challenges can be grouped by focusing on data, model and service.

The **data challenges** are related to four main dimensions: acquisition, integration, transmission and security. During the acquisition, the initial focus is on the accurate and efficient collection of data from built assets. The integration of these sensors into existing structures requires careful planning in terms of positioning, powering devices, access for maintenance and vandalism prevention amongst other considerations. Those devices must establish clear and uninterrupted connections with a communication network, complicating integration at the onset, and potentially O&M later on. (Dave et al 2018) Those connections must support potentially large data throughputs, which can put a strain on existing network infrastructures. A lack of expertise, namely related to choosing the right technologies for subsequent phases and meet built asset service life expectations can also constitute a challenge. Once the data is properly transmitted, it must be stored. Towards seamless integration, data interoperability and data fusion must be achieved to ensure that data from a variety of sources (e.g. sensors, systems, external) particularly in a heterogeneous systems environment, are successfully used. (Khajavi et al 2019) It also is critical that all data related matters must be dealt with security and privacy in mind (Shahzad et at 2022).

The **model challenges** concern the creation of a dynamic model that can evolve according to changes in the physical asset and new user requirements. It involves sophisticated architectural considerations to ensure that the model faithfully reflects the built asset. Interoperability issues may need to be addressed again here, alongside the development of standardized classifications and structures. (Shahzad et al 2022)

Closer to users, **service challenges** involve defining services and their requirements. Common challenges revolve around poorly defined user requirements, leading to wasted efforts and delays. Services often suffer from a lack of foresight relating to scalability, maintenance and resource management, resulting in inefficient and inflexible systems.

Finally, beyond those known challenges in the built asset community, many obstacles typically met in software engineering have yet to surface.

2.3 DevOps

Over the last decade, DevOps has emerged as the prominent approach to increase agility, productivity, and system quality in the software industry. It has resulted in spectacular progress regarding key aspects of software development and operations (Forsgren et al 2018). (Kim et al 2021) defines the three ways of DevOps: 1- Flow, 2- Feedback and 3- Continual Learning and Experimentation. DevOps is based on the principle of continuous improvement and focuses on optimizing the flow of activities involved in the creation of end user value, from idea to deployed functionality, and on providing fast and continuous feedback throughout the entire process. It extends the Lean and software agile methodologies by integrating aspects like quality assurance, continuous integration and deployment, and system operations together with development to enable more frequent and reliable releases. It aims to use automation as much as possible to improve productivity, predictability and quality.

Figure 2 illustrates the infinite loop that describes the DevOps workflow that is composed of the following phases: Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor. These phases can be grouped in four main process parts: Planning, which relates to the Plan phase; Dev, which includes the Code, Build, and Test phases; Release & Deployment, which includes the Release and Deploy phases; and Operation & Monitoring, which includes the Operate and Monitor phases.

The last part of the DevOps workflow, Operation & Monitoring, aims at ensuring the proper operation of the DT and the real-time monitoring of different aspects of its execution to enable early detection of issues (before they become a problem) and the identification of aspects that need to be improved. This part of the process rely on DT telemetry to collect the required data. It is important to mention that the Monitor phase provides a main input into the Plan phase and

allows closing the DevOps (infinite) loop that supports continuous improvement. Because of the space limitation of the paper, the technical aspects related to the Operate and Monitor phases, including their automation and tool support, will not be further discussed.

Figure 2. DevOps workflow

2.4 Related work

The development of digital twins has reached an important milestone, evolving from simple 3D models to integrated systems that harness real-time data, predictive analytics and machine learning (Li et al 2024). However, studies in built asset are still emerging and the body of work remains sparse. BIM plays a key role in this progression, improving the accuracy and efficiency of DT applications, marking a shift from traditional practices to advanced, data-driven methodologies. This underscores the importance of the growing development of long-term DTs in this field. Also, with data often at the heart of DT development challenges in the built environment, the lack of structure/framework around it is readily apparent. We have gathered here the most relevant studies to this paper.

The study on the development of a DT at the University of Cambridge (UK) describes a dynamic DT at the building level that integrates various heterogeneous data sources, such as a multi-layered BIM and IoT-based sensors. The system is built on a comprehensive system architecture aimed at "[…] supporting intelligent asset management, providing effective O&M management, and further bridge the gap between human relationships with buildings/cities via more intelligent, visual and sustainable channels." (Qiuchen Lu et al 2019). Despite the system approach, a framework designed to allow continuous updating and improvement to support long-term development is required.

(López-Peña et al 2020) worked on "fast and continuous monitoring feedback of system availability" activity (F&CF availability). This leverages the DevOps contribution to a component of DTs in IoT system from an availability standpoint. This approach materializes through the deployment of distributed components monitoring, which provides the DevOps team with the flexibility to configure specific metrics and indicators at runtime, thereby enabling customized, on-demand monitoring.

(Chamari et al 2023) proposes a service-oriented architecture (SOA) for data-driven applications. It focuses on component modularity and reusability as well as real-time integration of heterogeneous data sources. While this study does not explicitly speak to the evolution of DT over time, the SOA lays the foundation for a systematic evolution of DT.

Regarding DevOps, some papers have discussed the use of its principles for the development, deployment, and operations of DTs (Hugues et al 2020, Ugarte Querejeta et al 2020, and Dobaj et al 2022) and Cyber-Physical Systems (CPS) (Combemale & Wimmer 2020, Mertens & Denil 2022), but, to the best of our knowledge, there exist no DevOps approach or framework to support the systematic development and evolution of DTs.

In summary, the lack of structure in DT development for built assets appears in the various studies on the subject. This would be consistent with the experimental nature of the studies. Furthermore, the studies mostly concentrate on the improvements on single aspects of a DT. A broader and more systematic integration of DevOps aims at addressing the identified challenges and providing support for the systematic and scalable development and evolution DTs.

3 DevOps approach for DT

From an engineering perspective, DTs are generally considered as complex distributed software systems. As such, they need to be designed, developed, tested, deployed, and operated as any complex software system. For this purpose, the development of DTs needs to be well-planned and conducted in an iterative manner to support its systematic evolution and continuous improvement. From a functional perspective, the number of aspects addressed by a DT and the set services it provides need to be incrementally developed.

In this section, we describe a DevOps approach that can be used to support the systematic development and evolution of DTs in the context of build assets. We focus on three aspects: software methodology to support the iterative development of DTs; 2- Microservice architecture to enable the independent development and composition of different aspects of DTs; and 3- DevOps infrastructure to automate the continuous integration and deployment (CI/CD) of DTs.

3.1 Software Methodology for DT

From a software methodology perspective, the overall development process needs to be structured to support the systematic and iterative development and evolution of DTs so that they can be adapted to continuously changing environments, requirements, and needs. Throughout their lifecycle, besides the changes made to correct defects or refactor part of the software to reduce the technical debt, a DT may need to be modified for three main reasons:

- Modifications made to the AT or its environment that need to be reflected in the DT, e.g. structural properties of the AT have been modified, new sensors have been added or a new source of external data has become available.
- Modifications made to the DT to improve the precision of one of its existing services, e.g. a DT can be modified to take advantage of a new source of data or to introduce of a new type of model to improve its current management of a specific aspect (e.g. Thermal Comfort)
- Increase the set of services provided by the DT, e.g. the scope of a DT that was initially developed to improve comfort in a building can be modified to include a new aspect and services like the reduction of energy consumption, the improvement of air quality, or the management of assets.

For this purpose, each iteration needs to be carefully planned in the Plan phase of the DevOps workflow. To support the management of the software development, DevOps promotes the use of agile principles like the use of Kanban boards (or other types of sprint planning boards). Among other things, the planning of an iteration requires the determination of the percentage of resources that will be allocated to the development of new features/services, the improvement of existing ones, the correction of software defects, and the reduction of the technical debt (Kersten 2018). In this phase, the information collected by the DT telemetry in the Monitor phase plays a key role.

In the case of the GRIDD Lab, we started by focusing on the Thermal Comfort aspect, as illustrated in Figure 1. We first developed the 3D Visualization service and then added a Comfort Analysis service using on a PMV model. Each service was independently developed and then integrated in the DT. In a second phase, we increased the scope of the DT by adding an Asset Management aspect, which includes two main services: Equipment Inventory and Room Configuration. Figure 3 provides a high-level architecture diagram of the GRIDD Lab DT software architecture resulting from this new aspect. As the DT continues to evolve, additional aspects such as energy consumption management and air quality improvement will be iteratively incorporated.

3.2 DT Architecture - Microservices

DevOps promotes the use of a microservice architecture to manage the software complexity of DTs. Some existing DTs, while complex, are structured with most of the code running on one machine, gathering and processing data, akin to monolithic style architecture. This is easier to understand but challenging to scale when adding new features. Some others employ distributed systems that use cloud services with some degree of decoupling. While the latter are closer to the

proposed architecture, this section examines what they are and how they could positively impact DTs in built assets using GRIDD's Lab example.

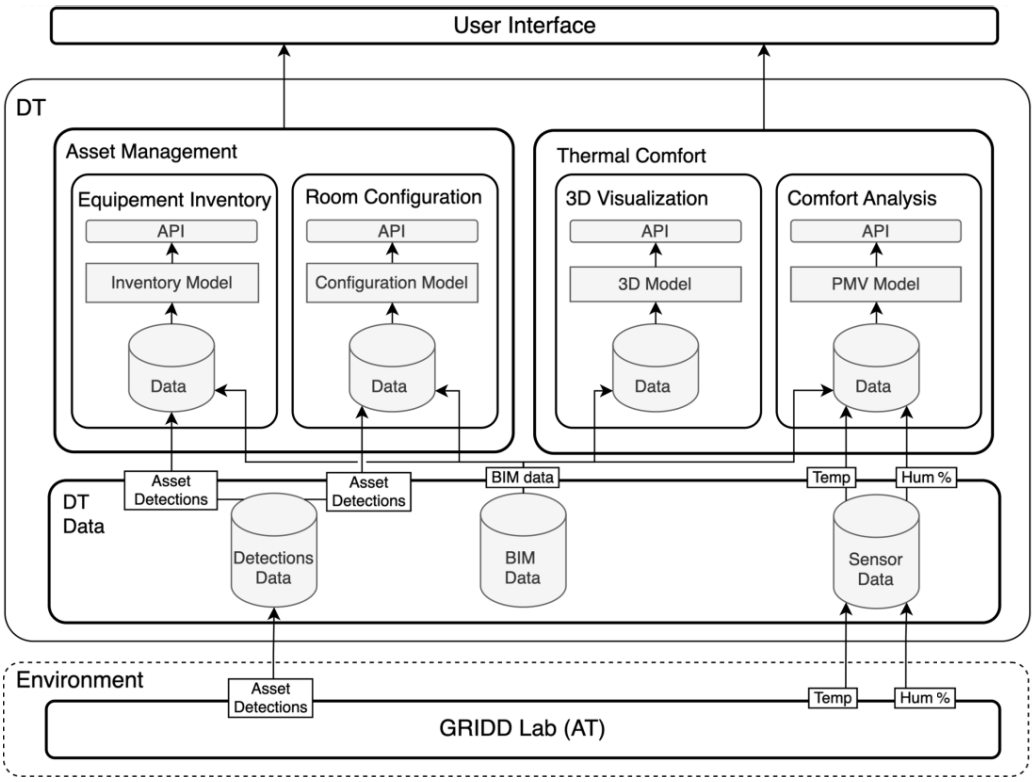


Figure 3. Software architecture of the GRIDD Lab DT after the addition of Asset Management

First, independent deployability is considered. This principle asserts that a microservice can be changed and deployed without affecting other services. Achieving this requires ensuring that microservices are loosely coupled yet maintain stable interfaces, enabling flexibility, faster release cycles, and simplified deployments across diverse platforms and environments. For instance, in Figure 3, Thermal Comfort aspect could be updated to improve performances without impacting Asset Management services. The User Interface would remain unimpacted if the API stays the same. This helps to address model and service challenges.

Second, models are developed around business domains to decompose the DT system into a set of components based on real-world domains they operate in. Each DT is associated with a domain of expertise, one team and one microservice. Splitting a larger DT into smaller components aligns with the Conway Law as domain-specific teams, such as those working on equipment inventory, are unlikely to collaborate with unrelated teams, such as those focused on PMV. Those are two separate domains, heterogenous both in model and data, thus likely to communicate very differently. In our example, it is likely that any aspect added to the GRIDD Lab DT will be by a different research team, following its own objectives and methodologies. Drawing clear boundaries helps organize around specific purposes and goals of a DT by keeping internal cohesion high and external coupling low. This helps to address some data and service challenges.

Third, state ownership is enforced. Each microservice must be able to independently manage its data, maintain its state and control what information is shared, exposing only the information that needs being seen. For instance, a Room Configuration component would store the current state of configuration, allowing it to check against a planned configuration and create work orders to change them. Another component or service could need this information, but would have to request it through the API. As depicted in Figure 3, this means having its own data to facilitate transformation of data while ensuring persistence. This abstraction typically spans several containers (e.g. Docker), each encapsulating everything needed to run a microservice in any

environment, regardless of the programming language, thereby standardizing deployment and simplifying the management of various service instances. This helps to address some data challenges.

Finally, by fragmenting a DT into manageable and autonomous services, microservices architecture offers a modularity and flexibility that promotes simpler maintenance, greater scalability and continuous innovation, while responding more effectively to the changing needs of complex environments. DevOps processes and their associated tools thrive in decoupled architectures. Beyond the benefits brought by independent deployability, business domain decomposition, and state ownership, DevOps provides a natural solution to address challenges of DT development in build assets.

3.3 DevOps infrastructure to support the Dev and Release & Deployment phases of DT

A main contribution of the DevOps approach comes from the automation of different aspects of the DT software process. The goal is to streamline and improve the efficiency of the overall software process by automating manual repetitive tasks using different types of software technologies and tools[4]. The elimination of these repetitive tasks, which from a Lean perspective constitutes a waste, allows development teams to focus on value-added activities such as developing new features, improving existing ones, or correcting software defects. The automation of these manual tasks also has the advantage of reducing human error and increasing the consistency of tasks performed, resulting in a significant improvement in the quality of the final product. This not only facilitates the ongoing evolution of the DT, but also enhances its maintainability.

In this paper, we concentrate on three main aspects of DevOps process that are used to reap the full benefits using existing technologies:

- Version control using Git to support the overall DevOps process
- Continuous Integration (CI) to automate the Dev phases
- Continuous Deployment (CD) to automate the Release & Deployment phases

Figure 4 illustrates CI and CD in the context of the overall DevOps process lifecycle.

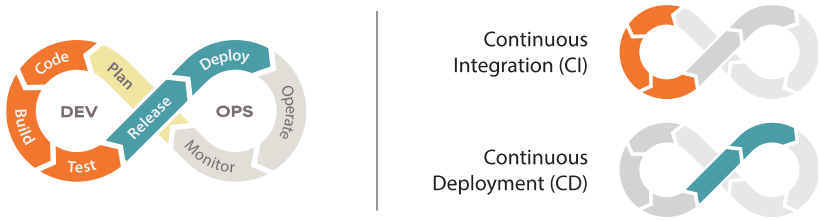


Figure 4. DevOps workflow – CI/CD phases

Version Control

In DevOps, Git[5] is used as a version control system to support efficient team collaboration and provide a single source of truth for all project artifacts, including code, test cases, configuration files, and documentation. It provides the foundation to support the overall DevOps process. It enables precise tracking of every modification made to a file, whether additions or deletions, and offers the flexibility to revert to previous states and compare file changes over time. It also provides support for file diff/merge, which allows teams to collaborate more efficiently on development projects.

Git is particularly relevant for offering robust branching models such as Gitflow, which organizes projects into two primary branches: Development and Master. This model streamlines workflows by allowing developers to work in dedicated development branches, often referred to as feature branches. Such a structured approach not only clarifies the development process, but

[4] A broad range of technologies and tools (open source and proprietary commercial) have been developed over the last two decades to support the different aspects of all phases of the DevOps process.
[5] Git: https://git-scm.com/

also integrates seamlessly into continuous integration and deployment (CI/CD) pipelines, supporting the iterative nature of modern software development.

Furthermore, Git platforms provide a set of basic functionalities to support different aspects of the DevOps process, including mechanisms for peer-review[6] and for the development of CI/CD pipelines[7]. These platforms also facilitate the creation and management of tasks/issues, incorporating different project management tools like Kanban boards to ensure a smooth, organized workflow across all stages.

In our project, the GitHub[8] platform is used to support our DevOps approach and GitHub Actions are used to implement all CI/CD pipelines. This solid infrastructure enables a smooth and secure transition of code from development to production. By properly exploiting Git, teams can maximize the benefits of CI/CD pipelines, taking full advantage of available tools and technologies.

Continuous Integration (CI)

In the Dev phases, different tools and technologies can be used to support and automate various aspects of the software development process. In particular, the CI pipeline automates the execution of the different steps involved in the Code, Build, and Test phases. This automation facilitates the seamless integration of new code into a shared Git repository (on Master on Main branch), where it is continuously tested to identify errors as early as possible to ensure that the latest software version is always in a state that is ready to be released to production.

Figure 5 outlines how each phase of the CI pipeline—Code, Build, and Test—is meticulously thought out to improve the efficiency across various activities, to support and automate various aspects of the software development process.

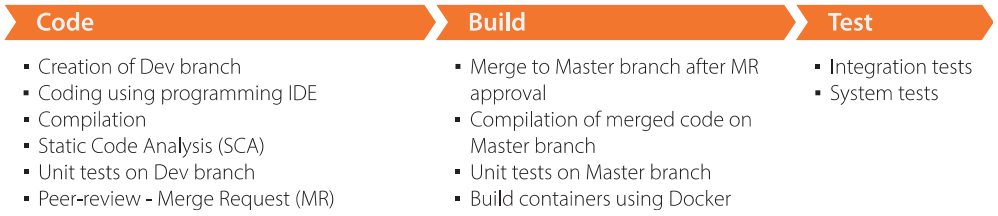


Figure 5. Typical CI pipeline tasks

Code: In this phase, software developers perform various tasks associated with the development of new functionalities, improvement of existing ones, or fixing of issues. Different tools[9] are used to support the coding tasks, including code editing (using programming Integrated Development Environments (IDE)), compilation, static code analysis, and unit tests. In this phase, the work is done on a local Git development branch, which allows developers to work independently from each other. The phase concludes with a code peer-review.

Build: The Build phase is triggered once the new code has been accepted by the reviewer. It starts with the integration (merge) of the new code on the Git Master branch. During this step, Git ensures that the code can be integrated and that no conflict arise with the existing code. Once the new code is integrated, the unit tests are executed on the Master branch and a new version of the microservices (affected by the code change) is build (e.g. as a Docker container). Finally, the new candidate version of the software is packaged and made available for testing.

Test: As soon as a new candidate version is made available (by the Build phase), the different integration and system test phases are automatically executed. The testing is conducted in a pre-production environment. The goal is to identify issues before the new version of the software is deployed in the pro-duction environment. In the context of DT for build assets, this typically

[6] For example, Pull Requests in GitHub and Bitbucket, and Merge Requests in GitLab.
[7] For example, GitHub Actions and GitLab CI.
[8] GitHub: https://github.com/
[9] Different programming languages are used for different components, including C++, Java, and Python. ESlint (eslint.org) and SonaQube (www.sonarsource.com) are the main tools used for static code analysis.

includes testing on specific devices and environments (e.g. IoT device or cloud environment) depending on the component (microservice).

In our projects, we use both Azure and AWS for different cloud services (e.g. Azure for hosting the main database that stores the data collected from the different sensors deployed in the room and AWS for the deployment of different microservices), and a combination of Arduino, Raspberry Pi, and Jetson devices for the deployment of different data acquisition components and edge-microservices that perform local treatment of data. Although the automation of deployment on cloud environments is nowadays common, the deployment on IoT devices is still often performed manually. As a result, a deployment pipeline needs to be developed for each of the environments on which components/containers are deployed, this includes Azure, AWS, Arduino, Raspberry Pi, and Jetson devices.

Continuous Deployment (CD)

In the Release & Deployment phases, CD pipelines automate the deployment of verified code in the production environment, while maintaining the stability of the DT system, thus ensuring rapid and secure updates. In a fully automated DevOps process, CD pipelines can be triggered whenever the product manager is ready to release a new version of the DT software in production. By effectively implementing CI pipelines and leveraging a microservices architecture, the transition to CD occurs naturally. The CI ensures a reliable source of code changes, setting the stages for these changes to be automatically deployed anytime through a streamlined process. This process uses Docker containers to provide an isolated and consistent environment for each component, making it easier to deploy, scale, and maintain the application across different operational environments—from development to production. CD pipelines enable a seamless transition from development to production, crucial in heterogeneous environments where both cloud infrastructure and edge devices, such as IoT devices, are used. This deployment flexibility is essential for maintaining the real-time operational integrity and scalability of DT systems, particularly in complex built asset management scenarios.

4 Conclusion

This paper proposes a DevOps approach to support the systematic development and evolution of DTs for built assets. The proposed approach is described in terms of (1) a software methodology to support the systematic and iterative development of DTs, (2) microservice architecture to enable the independent development and composition of different aspects of DTs, (3) DevOps infrastructure to support the development, integration, and deployment of DTs. Regarding the DevOps infrastructure, we focused on three main aspects: version control using Git to provide a single source of truth and support team collaboration, Continuous Integration (CI) to support the Dev phases, and Continuous Deployment CD to support the Release & Deployment phases.

The paper illustrates the approach using the example of the GRIDD Lab, effectively showcasing how DevOps can support iterative development, and enhance the capability of DTs to provide valuable insights and services such as comfort management and energy efficiency.

Besides providing a solid foundation for the development and operations of scalable DTs, the use of DevOps also allows addressing several of the main challenges associated with the development of DTs for build assets as described in section 2.2.

In conclusion, the proposed DevOps approach marks a significant shift towards more adaptive DT systems in the built environment. It not only fosters a culture of continuous improvement and collaboration but also sets a foundation for future research and development in this area. The development of a DevOps framework for DT to support the approach proposed in this paper is currently underway and will be published and released as open source in the coming months. The proposed approach continues to evolve as we work on the ongoing evolution of the GRIDD Lab DT and the development of new DTs.

Acknowledgements

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada.

References

Bolton, A., Butler, L., Dabson, I., Enzer, M., Evans, M., Fenemore, T., Harradence, F., Keaney, E., Kemp, A., Luck, A. and Pawsey, N. (2018). Gemini principles. https://doi.org/10.17863/CAM.32260.

Combemale, B. and Wimmer, M. (2020). Towards a model-based devops for cyber-physical systems. In Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6–8, 2019, Revised Selected Papers 2 (pp. 84-94). Springer International Publishing.

Camposano, J. C., Smolander, K., & Ruippo, T. (2021). Seven metaphors to understand digital twins of built assets. IEEE Access, 9, 27167-27181. https://doi.org/10.1109/ACCESS.2021.3058009

Chamari, L., Petrova, E. and Pauwels, P. (2023). An end-to-end implementation of a service-oriented architecture for data-driven smart buildings. IEEE Access.

Chenari, B., Dias Carrilho, J., & Gameiro da Silva, M. (2016). Towards sustainable, energy-efficient and healthy ventilation strategies in buildings: A review. Renewable and Sustainable Energy Reviews, 59, 1426-1447. https://doi.org/10.1016/j.rser.2016.01.074

Dobaj, J., Riel, A., Krug, T., Seidl, M., Macher, G., & Egretzberger, M. (2022). Towards digital twin-enabled DevOps for CPS providing architecture-based service adaptation & verification at runtime. In Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 132-143).

Dobaj, J., Riel, A., Macher, G. & Egretzberger, M. (2023). Towards DevOps for Cyber-Physical Systems (CPSs): Resilient Self-Adaptive Software for Sustainable Human-Centric Smart CPS Facilitated by Digital Twins. Machines, 11(10), p.973.

Dave, B., Buda A., Nurminen, A., and Främling, K., (2018). "A framework for integrating BIM and IoT through open standards," Automation in Construction, vol. 95, pp. 35–45, doi: 10.1016/j.autcon.2018.07.022.

Eramo R., Bordeleau F., Combemale B., et al. (2021). Conceptualizing digital twins. IEEE Software 39(2):39–46.

Fanger, P. O., Breum, N. O., & Jerking, E. (1977). Can colour and noise influence man's thermal comfort Ergonomics?, 20(1), 11-18.

Forsgren, N., Humble, J. and Kim, G. (2018). Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations. IT Revolution.

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In Springer International Publishing (pp. 85-113). https://doi.org/10.1007/978-3-319-38756-7_4

Halhoul Merabet, G., et al. (2021). Intelligent building control systems for thermal comfort and energy-efficiency: A systematic review of artificial intelligence-assisted techniques. Renewable and Sustainable Energy Reviews, 144, 110969. https://doi.org/10.1016/j.rser.2021.110969

Hugues, J., Hristosov, A., Hudak, J.J. & Yankel, J. (2020). TwinOps-DevOps meets model-based engineering and digital twins for the engineering of CPS. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 1-5).

Jones, D., Snider, C., Nassehi, A., Yon, J., & Hicks, B. (2020). Characterising the Digital Twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology, 29, 36–52. https://doi.org/10.1016/j.cirpj.2020.02.002

Kim G., Humble J., Debois P., et al. (2021). The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution.

Kersten, M. (2018). Project to product: How to survive and thrive in the age of digital disruption with the flow framework. IT Revolution.

Kritzinger W, Karner M, Traar G, et al (2018). Digital Twin in manufacturing: A categorical literature review and classification. Ifac-PapersOnline 51(11):1016–1022.

Khajavi, S. H. Motlagh, N. H. , Jaribion, A., Werner, L. C. , and Holmström, J. (2019). "Digital Twin: Vision, Benefits, Boundaries, and Creation for Buildings," IEEE Access, vol. 7, pp. 147406–147419, doi: 10.1109/ACCESS.2019.2946515.

Li W., Zhou H., Lu Z., & Kamarthi S. (2024). Navigating the Evolution of Digital Twins Research through Keyword Co-Occurence Network Analysis. Sensors, 24(4), p. 1202. Available at: https://doi.org/10.3390/s24041202.

Lo, C. K., Chen, C. H., & Zhong, R. Y. (2021). A review of digital twin in product design and development. Advanced Engineering Informatics, 48, 101297. https://doi.org/10.1016/j.aei.2021.101297

López-Peña, M. A., Díaz, J., Pérez, J. E., & Humanes, H. (2020). DevOps for IoT systems: Fast and continuous monitoring feedback of system availability. IEEE Internet of Things Journal, 7(10), 10695-10707.

Lu, Q., Vivi, et al. (2019). Developing a dynamic digital twin at a building level: Using Cambridge campus as case study. International Conference on Smart Infrastructure and Construction 2019 (ICSIC) Driving data-informed decision-making. ICE Publishing, 2019.

Mertens, J., & Denil, J. (2020). Digital twins for continuous deployment in model-based systems engineering of cyber-physical systems. In CEUR workshop proceedings (Vol. 2822, pp. 32-39).

Mertens, J., & Denil, J. (2021). The digital twin as a common knowledge base in devops to support continuous system evolution. In International Conference on Computer Safety, Reliability, and Security (pp. 158-170). Cham: Springer International Publishing.

Sharafdin, P. (2024). Building occupant comfort monitoring using IoT-based sensor data integrated into Digital Twin. M.Sc. Thesis, Ecole de technologie superieure (ETS), Montreal (Canada).

Shahzad, M., Shafiq, M.T., Douglas, D. and Kassem, M. (2022). Digital twins in built environments: an investigation of the characteristics, applications, and challenges. Buildings, 12(2), p.120.

Tao, F., Zhang, H., Liu, A., & Nee, A. Y. (2018). Digital twin in industry: State-of-the-art. IEEE Transactions on industrial informatics, 15(4), 2405-2415.

Ugarte Querejeta, M., Etxeberria, L. and Sagardui, G. (2020). September. Towards a devops approach in cyber physical production systems using digital twins. In International Conference on Computer Safety, Reliability, and Security (pp. 205-216). Cham: Springer International Publishing.

JuNo-OPS: A DevOps Framework for the Engineering of Digital Twins for Built Assets

Sara Aissat
sara.aissat.1@ens.etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

Jonathan Beaulieu
jonathan.beaulieu.2@ens.etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

Francis Bordeleau
francis.bordeleau@etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

Julien Gascon-Samson
julien.gascon-samson@etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

Ali Motamedi
ali.motamedi@etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

Érik Poirier
erik.poirier@etsmtl.ca
École de technologie supérieure
Montréal, Québec, Canada

ABSTRACT

Digital Twins (DT) constitute complex software systems that need to be continuously modified/updated to meet evolving user requirements and priorities, and support continual improvement. Because they aim at monitoring and improving different system aspects, their development requires the collaboration of people from different domains of expertise, e.g. the development of a DT for built assets may involve the collaboration of experts in software engineering, thermal comfort, air quality, and energy consumption, etc. Consequently, DTs need to be engineered to enable the fast and secure integration and deployment of new code, the systematic and iterative evolution of their components, and the independent development of different system aspects by those experts.

In this paper, we present JuNo-OPS, a DevOps framework for the engineering of DTs for built assets. The framework is being developed, tested and validated in the the context of a multi-function room at École de Technologie Supérieure (ETS). We focus on two main facets of the DT software: the microservices architecture; and the DevOps infrastructure used to support the development, continuous integration, continuous delivery and the automation thereof. We also discuss challenges and next steps related to the development and evolution of the framework.

CCS CONCEPTS

• **Software and its engineering** → *Publish-subscribe / event-based architectures*; **Cloud computing**; **Software infrastructure**; **Collaboration in software development**.

KEYWORDS

DevOps, Digital Twins, CI/CD Pipelines, microservices, IoT

ACM Reference Format:
Sara Aissat, Jonathan Beaulieu, Francis Bordeleau, Julien Gascon-Samson, Ali Motamedi, and Érik Poirier. 2024. JuNo-OPS: A DevOps Framework for the Engineering of Digital Twins for Built Assets. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3652620.3688266

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MODELS Companion '24, September 22–27, 2024, Linz, Austria
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688266

1 INTRODUCTION

A Digital Twin (DT) is a digital representation of an actual system[1], also called the Actual Twin (AT), that is dynamically and constantly updated with system data, and provides a set of services related to the system. In the context of construction engineering, DTs developed for the purpose of monitoring and improving/optimizing built assets can generally be viewed as IoT or Cyber-Physical Systems (CPS) that collect data using a broad range of sensors and APIs. Besides having to deal with the heterogeneous nature of these systems, the engineering of DTs developed in this context need to address the following three essential characteristics.

DTs are complex software systems.
DTs constitute complex software systems that need to be continuously modified/updated to meet evolving user requirements and priorities, and support continual improvement. As such, their development and evolution need to be supported by a software engineering process that allows for systematic incremental development/evolution and the automation of process tasks to enable fast and reliable delivery of new DT software versions.

DTs are associated with systems composed of multiple aspects.
The systems with which DTs are associated, i.e. the ATs, are composed of many different aspects. For example, in the domain of built assets, a building is associated with a broad range of aspects that include thermal comfort, energy consumption, air quality, asset management, security, and waste management. From an engineering perspective, a DT that aims to monitor and improve different aspects of a building needs to be incrementally developed based on a evolving set of concrete goals and use cases that are associated with the user/stakeholder requirements and priorities [1, 2].

[1]While definitions of DT traditionally refer to *Physical Systems*, we prefer to use the term *Actual System* because DTs can be associated with different types of systems that are not exclusively physical, including processes, Cyber-Physical Systems (CPS), and socio-technical systems.

The development of DTs is a multi-disciplinary process. It is fundamental to consider the fact that the development of DTs is a multi-disciplinary process that requires the collaboration of people/engineers with different expertise, which includes software engineers, but also domains experts in the different aspects addressed by the DT, e.g. thermal comfort, energy consumption, and air quality. The role and responsibilities of the software engineers are mainly related to the software architecture of the DT, the management of data (heterogeneous in nature), and the engineering process that is required to develop, test, deploy, operate, monitor, and evolve the DT software. The required domains depend on the type of systems associated with the DT, the purpose and goals of the DT, and the set of services that it provides. Experts are responsible for the definition of the metrics that need to be monitored and improved, the selection of the models and algorithms that are required to compute the metrics (using the DT data), analyse them, and provide specific DT services.

In software engineering, the Conway law [3] has established that there is a direct relationship between the software architecture and the organisation of the software teams. Consequently, the software architecture can be used to enable the independent development of the different software components of a system. Based on this, one of our objectives is to leverage the Conway law to allow domain experts to contribute to specific DT components without having to handle software engineering-related issues.

Different frameworks have started to emerge in the research community over the last years to address different aspects of DTs, including the evolution of DTs [4], DT composition [5], and and DT interoperability[6]. There also recent work on DevOps and DTs [7–13]. However, to the best of our knowledge, there is no DevOps framework to support the engineering of DTs.

This paper describes[2], a DevOps framework that has been developed over the last years to support the systematic development and evolution of DTs in the context of built assets. It directly addresses the three essential characteristics of DT previsouly described. The parts of the framework described in this paper have two main objectives: (1) support the systematic development/engineering of DTs and automate as many phases of the software process as possible to enable the fast and reliable deployment of software updates by all people involved in the DT engineering process, including non-experts in software engineering and DevOps; and (2) allow domain experts to contribute their expertise to specific aspects of the DT as easily as possible, i.e. without requiring detailed knowledge of the relationships with other system aspects and the details of the software engineering process, i.e. how to test, deploy, and how the software contributions are monitored.

The content of the JuNo-OPS is based on the experience acquired and the technologies developed in different projects related to DTs and DevOps, including: the development of DTs for different aspects of built assets and other application domains, research projects on the systematic improvement of industrial DevOps process (conducted in the context of the Kaloom-TELUS ETS Industrial Research Chair in DevOps), and the development of cloud-based solutions to monitor and analyse smart city open data. The current framework is implemented in GitLab, but different elements of the framework have also been implemented in GitHub in the context of different projects.

[2]JuNo is short for *jumeau numérique* (in French), which translates to *Digital Twin.*

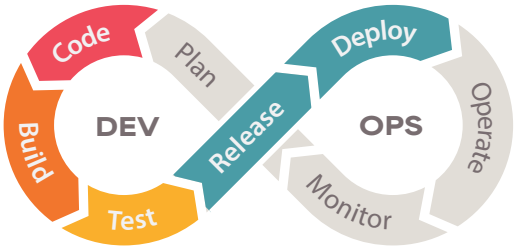


Figure 1: DevOps lifecycle loop

The main contributions of this paper are:

- Definition of a DevOps framework to support the engineering of DTs for built assets
- Microservices architecture for DT
- Description of challenges and lesson learned over the last years developing and deploying such DTs.

The rest of this paper is structured as follows. Section 2 introduces the main DevOps concepts that are directly relevant to the content of the paper. Section 3 describes the GRIDD lab that has been used to develop the core elements of the framework. Sections 4 and 5 describes respectively the DT microservices architecture and the DevOps infrastructure that constitute the core of the JuNo-OPS. Section 6 then discusses some key challenges that have to be considered in the design of a DT framework, and section 7 discusses the related work. Finally, section 8 concludes the paper and discusses future work.

2 DEVOPS BACKGROUND

DevOps [14] has emerged over the last decade in the software industry as the prominent approach to increase agility, productivity, and system quality. It has made a major impact on key metrics of the software process, including deployment frequency, mean lead time for changes, mean time to recovery, and change failure rate [15]. [14] describes the underpinning principles of DevOps as three ways: 1- Flow, 2- Feedback and 3- Continual Learning and Experimentation. Globally, DevOps aims at optimizing the flow of activities involved in the creation of end user value, from idea to deployed functionality, and on providing fast and continuous feedback throughout the entire process to enable continual learning and experimentation.

Figure 1 illustrates the DevOps lifecycle loop diagram as it is generally described in the literature. It is composed of a sequence of phases that include Plan, Code, Build Test, Deploy, Release, Operate, and Monitor. In this paper, we focus on a subset of these phases that go from Code to Deploy. Figure 2 illustrates that part of the process and identifies the set of main tasks that need to be executed during each phase. For example, the Code phase is described as the sequence of the following tasks: Code, Commit & Push, Static Analysis, Unit Test, Review, Merge to Main. Also, as indicated by the diagram legend, the Code, Commit & Push, Review, and Merge to Main tasks are manual, i.e., they explicitly require the involvement

of human resources, while all of the other tasks aims at being automated. Details of the tasks are provided in Section

Continuous Integration (CI) and Continuous Deployment (CD) play a key role at the core of the DevOps process [14]. Continuous Integration aims to integrate each developer's code with that of their peers following a change. The idea is to do this continuously to receive quick feedback, identify incompatible changes, and fix them as quickly as possible. This early detection of issues reduces the cost and effort required to fix them. To achieve this integration, it is crucial to ensure that the build is still functional. This way, it is possible to identify, for instance, when a change prevents the project from compiling or building. The next step is to run quick verification with different levels of testing such as static code analysis and unit tests. Automating these processes improves efficiency, allowing developers to focus on adding new features.

Continuous Deployment follows Continuous Integration. CD concludes with the creation of a package that can be delivered to production, and these packages are automatically installed on pre-production environment for further tests. This allows for verification and validation of production readiness and the decision on whether a specific set of changes can be delivered to production following a review process. This practice ensures the consistency and reliability of the code, thus increasing the overall quality of the software.

3 CONTEXT: GRIDD ENVIRONMENT

3.1 Description of GRIDD lab

The GRIDD (Groupe de Recherche en Intégration et Développement Durable en environnement bâti[3]) is an ETS (École de Technologie Supérieure) research laboratory focused on digital transformation, industrialisation, collaboration and sustainability. The research group is committed to driving sustainable change in the construction industry.

The GRIDD lab is a multi-functional room at ETS that is used for both research and teaching activities. The room is divided in six sections: one closed office space, two semi-private office spaces, two adjacent open meetings spaces, and a storage room. A 3D model screenshot of the lab is provided in Figure 3 [16].

Over the last years, a BIM model of the room has been created and different sensors have been developed and installed in the GRIDD lab in the context of different DT research projects. In particular, [16] has developed a plug-and-play IoT sensor infrastructure that has enabled the deployment of different types of sensors. As a result, the lab is currently equipped with a network of sensors that are used for collecting data at regular intervals, including temperature, humidity, and noise level. The lab is also equipped with two cameras: one located at the door to monitor the entries and exits in the lab, and one that is responsible for taking pictures that are used to monitor the count of specific elements (e.g. tables, chairs, and boards) present in the space and the configuration of the space in terms of the location of the tables, chairs, and boards present in the room (privacy requirements are taken into consideration).

[3]The english translation would be Research Group in Integration and Sustainble Development in the Built Environment

3.2 Digital Twin Projects Developed within the GRIDD Lab

The GRIDD lab has been the source of multiple DT projects that were developed with the aim of enhancing various aspects of its functionality and management. These projects have contributed to laying the groundwork for integrating different components into a cohesive DT system, as well as experimenting with various technologies for the representation of 3-dimensional spaces.

Thermal Comfort Digital Twin
We initially developed the Thermal Comfort DT. The goal of the project was to enhance the environmental conditions for the participants by monitoring and controlling factors such as temperature and humidity. Using the existing network of sensors, the project collects real-time data, which is analyzed and used to adjust the lab's environmental settings to optimize comfort for its users. The project included the creation of a 3D representation model and using Unreal Engine to visualize the data and the effects of various environmental adjustments.

Asset Management Digital Twin
The Asset Managment follow-up project focused on real-time inventory monitoring and spatial tracking of objects within the GRIDD lab. Using advanced camera systems and sensors, this project captures and updates information about the types and quantities of objects in the lab (e.g., chairs, tables, computers, participants), ensuring accurate resource management and planning. The data collected is visualized in a 3D virtual environment using Unity engine ™, allowing stakeholders to maintain a precise understanding of inventory levels and object locations, both at the present time and using historical data.

3.3 DT-based Projects outside the GRIDD

We have laid the groundwork in developing other DT-based projects. One point that we have looked at is the visualization of various DT services information into dashboards (e.g., Grafana) to provide a unified view of telemetry data. We have also explored the idea of macro-scale DTs, for instance, we tackled the challenge of modelling a large city-wide fleet of vehicles that are in motion and in which we extract high-level aggregated insights and visualizations.

4 DT MICROSERVICES ARCHITECTURE

DevOps processes are better implemented on loosely coupled architectures [14, 15]. This helps manage the software complexity of DTs as we will see later in this section. In JuNo-OPS, we use a microservice architecture. Figure 4 is used as a reference to express the architecture throughout this section. In the sub-sections, details are given on its components and how they are implemented in the context of the GRIDD projects to realise their potential benefits, as well as the reasoning behind our design choices.

4.1 The AT (GRIDD Lab)

Collecting the data in the AT is made possible by a network of sensors, which are distributed components with varying level of logic at the edge. Building a DT for built assets implies that not all

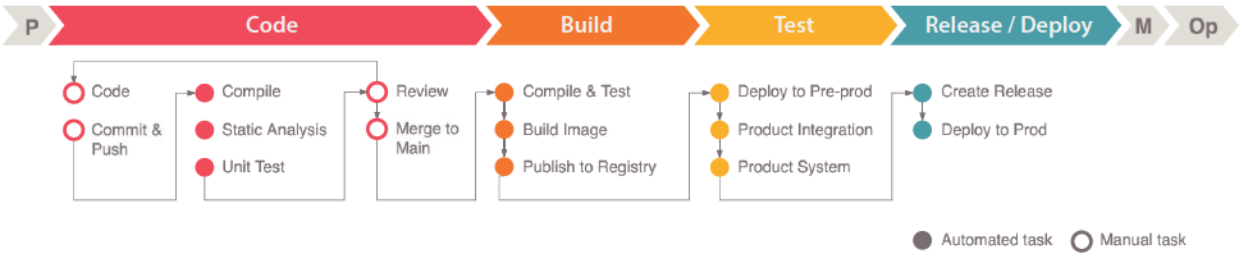


Figure 2: DevOps process: from Code to Deploy

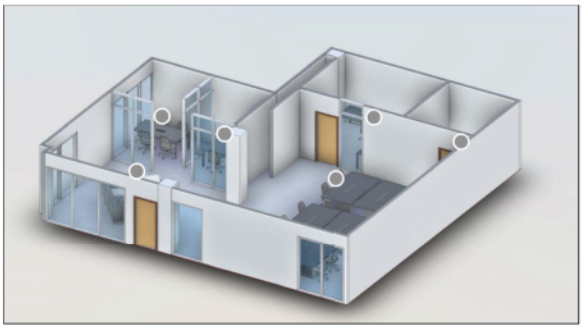


Figure 3: 3D virtual view of the GRIDD Lab

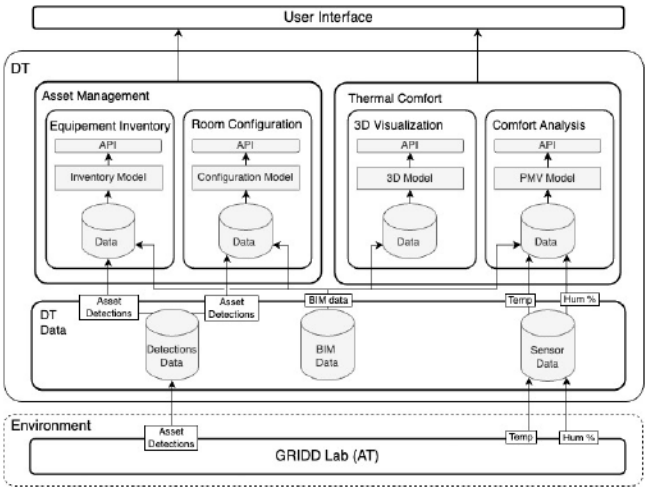


Figure 4: Software architecture of the GRIDD Lab DT after integrating the Asset Management and Thermal Comfort DTs

of the processing or storage will take place on a server or resource-rich cloud. While not strictly impossible, various constraints led us to take the design decision to offload some of the processing at the edge. For example, sending all of the raw images to the cloud for processing can use a large amount of bandwidth, which the local network might not be able to sustain, and which could also be cost prohibitive.

At the edge layer, the GRIDD Lab DT involves Internet of Things (IoT) technologies, with several resource-constrained devices, such as Single Board Computers (SBC), which run a familiar OS (e.g., a Linux variant), but are more resource-limited, or simple micro-controller (µC) OSes (e.g., RTOS). In the case of Linux-based SBCs, we deploy a lightweight Docker image including relevant Python dependencies. The Asset Management DT uses a Nvidia Jetson ™ device that includes a custom GPU, runs Ubuntu™ Linux, and includes computer vision modules such as `PyTorch`, `Ultralytics` and `PyRealSense2` to process camera images at the edge, so that only the extracted relevant Deep Learning Model detection results are pushed upstream, rather than uploading the raw images to the cloud. On the other end, the Thermal Comfort DT uses a Raspberry Pi ™ device (RPI) as a Bluetooth Low Energy ™ (BLE) hub, and includes the `Bleak` Python module. Several µC Arduino™-based sensor boxes are connected to this hub. When a new version of the firmware is produced, these devices must be updated using Over the Air Device Firmware Update (OTA DFU). In this case, each firmware build (e.g., a Docker image) would become an artefact checked in version control and deployed through the GitLab CI pipeline.

4.2 DT Data abstraction

The DT data abstraction manages and stores the data that the DT needs to accomplish its purpose. This abstraction is generic enough to include both static and dynamic data, as well as transient and permanent data, and is not tied to a given storage database or technology. Further, the data itself can be stored at various layers (e.g., cloud, edge). It contains data gathered from the environment (i.e., from the AT), as well as external data sources (i.e. Weather service, BIM server API). It is also the mediator between the AT and the DT as they only know about each other through data and events (e.g. pub/sub services). This is covered in more details in subsection 4.5 Communications.

The choice of the specific technologies to use, as well as *where* the data is stored, depend on the specific DT or combination of DTs that are modelled. In the context of the GRIDD DT, the data abstraction made of several independent DBs from one another, which accounts for the heterogeneity of the data types and also promotes looser coupling. For instance, JSON text documents are stored on an Atlas-hosted MangoDB instance on AWS for the asset detection data (Asset Management DT). Azure TSI hosts time series of temperature and humidity data. The 3D model BIM data is a COTS service from Autodesk.

Although the architecture is ready for independent "State Ownership" with a DB for each micro-service, it has not been applied yet since our DTs are only monitoring the AT; i.e., no actuators or connector to Building Management System (BMS) are currently in place to change any state in the AT (e.g., thermostat set-point or work orders for maintenance staff to alter the room assets). This integration will be the subject of future work.

4.3 Microservices Components

Generally speaking, each developed microservice is independent, and executes in isolation. Each microservice models a part of a DT (e.g., in Figure 4, the Equipment Inventory and Room Configuration services in the Asset Management DT). They are developed by different teams modelled around "business domains" (e.g., building occupant comfort, computer vision, BIM management). This is a conscious choice as it takes advantage of the natural tendency for each team to develop its part of a solution within a larger organisation, as described by the Conway Law. This keeps internal cohesion to a maximum while ideally keeping external coupling to a minimum. Those are highly desirable traits to maintain independent deployability and support automation, which are both in turn very important in a DevOps approach. Docker containers are used to run software components on the cloud and at the edge whenever possible. This forms an easily deployable build artefact which can also be checked in GitLab repo for version control. By allowing independent deployability of smaller yet complex functional components, we tackle the DT's complexity and multiplicity of aspects more easily.

4.4 User Interface (UI)

At the moment, each microservice has its own UI. The UI stack chosen by each team varies in the details, but revolves around dashboards such as Grafana and PowerBI™, complemented with 3D model visualisations to locate the information in space using the Unity Engine or the Unreal ™ Engine. Limiting the supported UI stack by the framework is considered to promote the reuse of code and services.

4.5 Communications

To promote a lower coupling, we avoid proprietary or component-specific protocols. Rather, the remote communications between the components takes place over RESTful APIs, using JSON messages. In the case of the Asset Management DT, the communications are managed by an AWS API Gateway and dispatched to an AWS Function-as-a-Service (FaaS) Lambda function for further processing and storage. In the case of the Thermal Comfort DT, the communications are brokered by an Azure IoT Gateway. For the front end servicing the UI layer, each micro-service has its own API that handles GET requests.

To better illustrate our design choices, we provide additional details on how the communications are handled concretely in the Asset Management DT (Figure 5).

Between the AT and the DT:

(1) A scheduled "snapshot" is sent from the Jetson device, after processing a video frame to detect the relevant objects (the Jetson sends a POST request)
(2) The Jetson's JSON data is transformed by the "insertSnapshot" lambda function, who adds an `id` field and a timestamp for indexing in Mongo DB
(3) The document is stored in the Altas cloud-based DB service on AWS

Between the DT micro-services and the UI:

(1) The scheduled latest "snapshot" is requested by the Unity Engine (EU), through a GET request
(2) The `getSnapshot` lambda function is activated by the API gateway to return the latest JSON document in the DB.
(3) Upon reception of the response, the view is updated to refresh the latest positions of the objects in the GRIDD lab

Figure 5: Asset Management DT Communications – Example

Overall, the DT microservices architecture serves the purpose of the DT while allowing evolution. It facilitates the implementation of DevOps processes which effectively manages the complexity of DTs in the case of the GRIDD lab. Being loosely coupled with well defined team boundaries and strong internal cohesion, the framework addresses the multi-disciplinary nature of DT while enhancing scalability, deployability, and automation.

5 DEVOPS INFRASTRUCTURE

One of the goal of the DevOps infrastructure is to automate as many phases of the overall software engineering process as possible. By streamlining repetitive tasks and improving the efficiency of the overall software process using a variety of technologies and software tools, the framework enhances productivity, consistency and quality, providing all the typical benefits of DevOps, including fast feedback and continuous learning.

This supports the seamless integration and management of heterogeneous components, such as IoT devices and external sources protocols or cloud services, which are essential for the dynamic and evolving nature of a DT. The automation provided by the framework benefits everyone involved in the DT engineering process, including software engineers who are developing code for specific DT platform components and domain experts, with no software engineering background, who can contribute their expertise to the core functionality of the DT without being constrained by software engineering details.

In this section, we describe three main elements of the DevOps framework: (1) the structuring of Git repositories to support the development of microservices-based (multi-aspect) DTs, (2) the CI pipelines and (3) the CD pipelines. The current version is implemented in GitLab and will be released in open source (under an MIT license) during the Fall 2024.

5.1 GitLab Repository Structure

The GitLab structure used to organize the various microservices can be viewed hierarchically, from the lowest level of an individual microservice to the highest level of the entire digital twin. The different microservices are grouped as sub-modules within functional systems, which are themselves sub-modules of the overall DT.

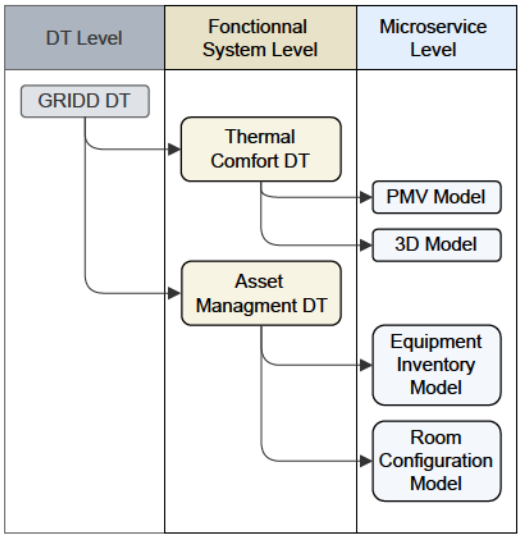


Figure 6: Hierarchical Structure of GitLab Repositories for GRIDD DT

Taking the example of the GRIDD DT as shown in Figure 6, the microservices related to Thermal Comfort DT, such as the PVM model and the 3D model, are grouped as sub-modules within this functional system. Additionally, the Asset Management DT, which also includes its own microservices, is grouped along with Thermal Comfort DT as another sub-module to form the complete DT of the main repository of the super project, GRIDD DT.

Git Workflows for Effective Digital Twin Development

Complementary workflows are used to support DT development across the different levels of the GitLab structure. A Git workflow is a process structured by a branching model that defines how code changes are introduced, reviewed and integrated into a Git repository, along with the various steps required to deploy the application to production. These branching models are used to manage feature development, testing, and deployment in an organized and collaborative manner at both the Microservice Level and the Functional System Level, as illustrated in Figure 7.

At the Microservice Level, GitLab Flow is established to simplify the rapid integration of new functionalities, divided as follows:

- **Main branch**: The Main branch serves as the collaborative base where all contributions from developers are integrated. The final changes are merged here, triggering the next level repository for subsequent actions. For example, a confirmed change in the PVM Model repository will trigger updates in the Thermal Comfort DT repository.
- **Feature branches**: Local branches, originating from Main, are created and used by developers to develop new features or modify existing ones. These branches allow for isolated development or bug fixes before the changes are merged back into the Main branch. For example, developers might create a feature branch to enhance a specific functionality or resolve an issue, ensuring that their work does not interfere with the main codebase until it is ready for integration.

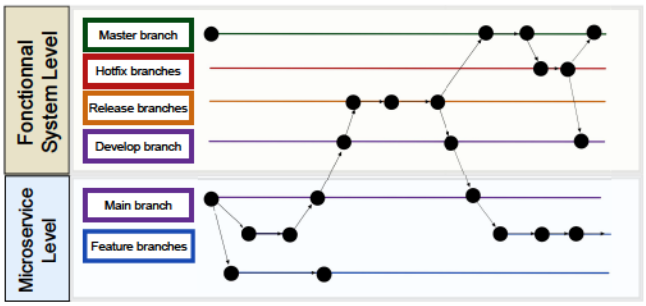


Figure 7: Branching Strategy for Multi-Level Development Workflows

At the Functional System Level, GitFlow was chosen to simplify the efficient management of deployment pipelines for different microservices while eventually ensuring seamless integration. The workflow at this level is divided as follows:

- **Develop branch**: The Develop branch serves as the integration branch for functionalities, incorporating the latest development work of the various microservices in preparation for deployment.
- **Release branches**: In these branches, the final testing is conducted in a production-like environment, and the bug fixes and urgent necessary tasks are performed (with no new feature development allowed). Once all of the bugs identified during the final test cycles are resolved, the release branch is merged into both the Master and Develop branches to ensure consistency.
- **Master branch**: The Master branch contains stable, production-ready code that has passed all necessary tests and reviews.
- **Hotfix branches**: The Hotfix branches are used for urgent corrections to the Master branch. These fixes are then merged back into both the Master and Develop branches to maintain consistency.

GitLab Merge Request

The Merge Request (MR) is the mechanism used in GitLab to ensure that any code modification is peer-reviewed before the code is accepted to be merged and integrated into the collaborative codebase. In a DevOps process, the creation of a merge request is automatically triggered whenever a developer submits a code modification that has successfully passed through the development pipeline, which will be discussed below.

Utilizing GitLab MRs within a branching model-based structure ensures controlled development and maintains code quality, keeping the Main branch always ready for production. At the microservice level, developers work on their local Feature branches, and with push permissions on the Main branch set to "no one", once ready, they are forced to submit their changes through the merge request process. This approach guarantees that only verified and validated changes reach the Release branch for further testing in a pre-production environment before being merged into the Main branch, facilitating a smooth deployment to production. Therefore,

to support continuous deployment and delivery, it is crucial to keep the Main branch always functional.

Finally, by setting up a GitLab structure and using workflows specific to each repository, the management of the various CI/CD pipelines becomes clearer and easier to implant at the different levels up to the complete DT.

5.2 CI Pipelines

Continuous Integration (CI) is composed of two main pipelines at the Microservice Level. A Dev pipeline, that is executed in the Code phase, followed by a Build pipeline.

The goal of the Dev pipeline is to ensure the quality of the code before it is reviewed (in the Review task). It is triggered each time a developer makes a commit or a push and runs on the developer's local branch at the microservice level. It automates the execution of the Compile, Static Analysis, and Unit Test tasks, each with the following purposes :

- **Compile**: This task performs the compilation of the code to ensure there are no syntax errors or other issues, and produce the executable.
- **Static Analysis**: This task uses static code analysis tools to check the quality of the code and its adherence to coding standards and conventions.
- **Unit Test**: This task automates the execution of the set of unit tests associated with the code that has been modified. It aims at verifying that individual components function as expected.

The goal of the Build pipeline is to package the microservice and publish its artifact in a binary[4] registry. It is triggered by the merge of the submitted code into the Main branch following its acceptance by the reviewer(s) during the peer-review process (Review task). The sequence of tasks executed in the Build pipeline is as follows:

- **Compile & Test**: The submitted code is first integrated with the rest of the codebase on the Main branch to ensure there are no integration conflicts. Although the code has already been compiled and tested on the developer's local feature branch in the Dev pipeline, it must be recompiled and retested (Unit Test task) after being merged into the Main branch to ensure that no problems have been introduced by the integration. Static code analysis (Static Analysis task) on the other hand doesn't have to be re-executed since the code itself has not changed.
- **Build Image**: In this task, the Docker image associated with the modified microservice is created. For this purpose, a base image must first be selected (from an existing library of Docker images), considering factors such as performance, footprint, security, and the source/provider of the image. The Docker file is then written using the selected base image, to which the microservice's source code and dependencies are added to finally build the image by running the docker build command.
- **Publish to Registry**: The new Docker image is then uploaded and stored in a binary registry, e.g. DockerHub[5] or Artifactory[6].

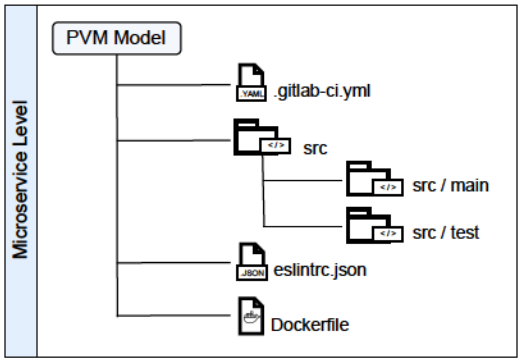


Figure 8: PMV Model microservice repo structure

Figure 8 uses the PVM Model as an example to illustrate the structure and content of the GitLab repository that is used to support execution of the two CI pipelines (i.e. Dev and Build pipelines). It contains the following elements:

- A **GitLab CI configuration file** called `.gitlab-ci.yml` that specifies the various stages of the CI pipeline at the root of the repository. This file is located at the root of the repository and contains different triggers to distinguish between the development pipeline and the microservice build pipeline and is located.
- The **source code directory** (`src`) containing two subdirectories: `src/main`, which holds the code of the microservice, and `src/test`, which includes the set of unit tests for the microservice that are executed as part of the pipeline.
- A **static code analysis configuration file**, in this case the `.eslintrc.json` file that is used by the ESLint [7] static code analysis tool. This file provides the set of rules to be applied to the repository's specific language as part of the Static Analysis task.
- A **`Dockerfile`**, that contains a series of instructions, such as specifying the base image, copying files, and running commands, to automate the process of building the Docker image for the microservice. This file is located at the root of the repository.

The pipeline thus ends by pushing the docker image in a registry, which will act as our artifactory where we store our versioned artifacts. This process then triggers the next-level pipeline, i.e. the repository that groups the microservices of the same functional system.

At this second level, an integration pipeline must be established to manage mixed deployments. Given that various microservices of a functional system are interconnected, it is crucial to perform an integration that respects these interdependencies. However, in

[4]The term binary refers to compiled or executable versions of software, such as executable files, libraries, or compiled code, that are necessary for running applications

[5]https://hub.docker.com/

[6]Artifactory is a DevOps solution for managing and distributing software binaries and artifacts

[7]https://eslint.org/

the case of the GRIDD digital twin, this aspect has not yet been addressed as the microservices are currently orthogonal.

5.3 CD Pipelines

The Continuous Deployment (CD) pipelines aims to automate the deployment of software systems and microservices in the different pre-production and production environments, where they can be executed, tested, and released.

We identify two main types of CD pipelines: Functional System deployment pipelines, that are responsible for deploying the overall microservices system, and Microservice deployment pipelines, that are responsible for deploying each individual microservice on its targeted execution environment, such as AWS cloud or specific IoT devices.

Functional System Deployment Pipeline
These pipelines take place at the Functional System Level. Once the code is ready in the Develop branch, it is sent to the GitFlow Release branch where comprehensive system-level tests such as end-to-end testing, user acceptance testing, and many more are performed. This is crucial for the eventual deployment of the digital twin as a whole, this time integrating its various functional systems into the top-level deployment pipeline, as will need to be done for microservices. When it is time to deliver the artifact to production, a specific artifact version is selected for release. This selection triggers the same CD process, but against the live environment, ensuring that the deployment is tested and validated under real-world conditions.

To facilitate continuous deployment in containerized environments, tools such as Kubernetes[8] and Helm[9] can be used to customize scalability. Kubernetes orchestrates containerized applications across a cluster of machines, ensuring efficient resource allocation based on application needs. Helm charts further simplify the management of deployments, providing templates for Kubernetes resources. This includes a new suite of tests that have yet to be fully implemented, further enhancing the robustness and reliability of the deployment process.

The Functional System deployment pipelines have not yet been implemented in the DevOps framework.

Microservice Deployment Pipeline
Regarding Microservice deployment pipelines, the goal is to develop a library of pipelines that supports all of the different nodes/targets that are required/used in the pre-production or production environment. This process begins by pulling the artifact from the artifactory and delivering it to one or more environments as necessary.

At this point, we have developed pipelines for AWS and Azure cloud environments, and for Raspberry Pi, Jetson, and Arduino IoT devices.

Deployment on Edge Devices
Building automated deployment pipelines for edge devices is a distinct use case with specific challenges. In implementing such pipelines, one challenge was met in the form of strict network administration rules. Direct access to a connected device through a standard SSH session (using agentless Ansible for automation) was not allowed by the firewall and local area network parameters. Using a VNC connection mediated through a cloud-based VNC server was considered to circumvent the problem, but it is not an "automatable" solution. Therefore, GitLab Runner installed on the edge device itself solved the issue because it acts as an agent to establish communication from within with GitLab server. This allowed the set up of a fully automated CD pipeline using GitLab CI and Docker. Figure 9 describes the sequence of actions that were executed for the Nvidia Jetson running Jetson Linux based Ubuntu 20.04. The RPI devices steps are similar as they use Linux based Ubuntu 20.04 as well.

Edge device initial set up:
(1) Git clone the Object Detection Model (ODM) repository
(2) Install `Docker.io`
(3) Create a Docker file with `3.12-bookworm` Python image
- Include the ODM's Python modules dependencies (e.i. `NumPy`, `PyTorch`, `Ultralytics`, `PyRealSense2`, etc.) as requirements.
- Run `Python3 -m main.py` script
- Push the Docker image to Docker Hub

(4) Install a GitLab Runner instance following GitLab's guidelines
(5) Register the Runner on the device and on GitLab's platform

Writing `.gitlab-ci.yml` file to define pipeline configurations:
(1) Tags: "Jetson" to help identify this part of the pipeline
(2) Job1: Pull Git repository head
(3) Job2: Build the Docker image and push it to Docker hub
(4) Deploy stage: Pull the Docker image from Docker hub and run it.

Figure 9: Sequence of actions for deploying on the Nvidia Jetson

This approach is acceptable on Linux-based devices, it is not ideal for scaling as having one agent for each edge device makes the infrastructure heavier to maintain. Potential solution to this covered in section 7.1 since not all devices is expected to be power hungry Linux based.

6 LESSONS LEARNED AND CHALLENGES

This section discusses some of the main challenges that we encountered in the design of our DevOps DT framework.

6.1 Deployment on Distributed Heterogeneous Edge-Cloud Platforms

As seen in Section 5.3, while most components of the DT models that we developed were deployed on the cloud, some components were deployed at the edge. This deployment heterogeneity raised several challenges.

Communication Network
A major challenge that had to be solved is related to the fact that

[8] https://kubernetes.io/
[9] https://helm.sh/

the edge-based Jetson device was located behind a highly restricted network, complicating the automated deployment of updated software components. In our case, the problem was temporarily solved, but the solution may not apply everywhere. Besides, pulling the code and building the Docker image on the same runtime node may seem like unnecessary extra work, but to make sense out of this approach, we need to consider the following. Once the network access problem will be permanently solved, the GitLab Runner will likely run on a local VM or public Cloud. Just like the approach described in this paper, it will generate an artefact (Docker image), store it under version control to be tested and should deploy to many identical edge devices instances across the ETS campus/network. Any SBC running a Linux OS acting as edge device is expected to apply a similar approach to CD pipeline automation.

Handling communication losses is also a challenge. How to update the microservice onto an edge-based device in the event of a network failure, or in the event of a partial OTA DFU which could lead to device bricking? In the same manner, how to handle a failing device, or how to handle fluctuating networking conditions (e.g., WiFi or cellular).

Power Management
In an effort to lower the total power consumption of the devices, we may consider slim down distributions of Linux such as Ubuntu™ Core or Yocto. The pipeline for updating OSes, especially those implying a kernel recompilation, has yet to be defined and may prove to be a challenge. µC are more energy efficient, but more specialised firmware are sensitive to communication losses during update process.

We came to realize that the deployment process in itself needs to consider distributed systems-related challenges. On slim down versions of Linux on edge devices may be more energy efficient, but Docker containers use may become a challenge deploy and run on such OS distribution.

6.2 Domain Experts with Non-Software Engineering Backgrounds

As previously mentioned, the engineering of DT associated with a complex system requires the collaboration of a group of people with different background, work culture, and expertise. This has implications. Those will be covered in this section.

Git Training
Some degree of training on Git is necessary on the tool itself, but also on best practices of how to use it. Indeed, in the DevOps approach, the use of Git in a specific way (e.g. trunk based with short lived feature branches) can be challenging at times for developers themselves. This could lead to misunderstandings for non-developers and significantly impact the whole process.

Test Suite
Writing proper tests to validate the quality of one's code is a specialty. Some cultures favors that developers write their own tests rather then handing off the task to quality control team, but this already implies that the developer are software engineers. Furthermore, test suite must be maintain and renewed. This challenge may only be answered by a software engineer being embedded in each micro-service team.

6.3 Data Security and Privacy

An architecture that comprises several components at various layers (e.g., edge and cloud) must consider data security and privacy aspects. At the basic level, traditional secure communications mechanisms should be put in place, such as SSL and authentication – for instance, in our GRIDD DT, all communications use HTTPS. In addition, due to privacy implications, it might be advisable to process some sensitive data locally to reduce the exposure risk. In the case of the Asset Management DT, given that we run the image recognition algorithm at the edge, the raw images containing sensitive data are not transmitted over the network. Further, protecting, and ideally shielding the data sources (e.g., databases) is also important in our GRIDD DT, all databases (DBs) are accessed only through API endpoints (i.e., the external components cannot directly access the DBs). Finally, ensuring that the microservice endpoints themselves are secure is paramount; otherwise, this could lead to disastrous consequences, such as an attacker accessing private data of the DT, and even performing unauthorized/dangerous actions on the AT itself.

7 RELATED WORK

The concept of Digital Twins (DTs) has evolved significantly, particularly in the context of built assets, encompassing buildings and infrastructure. DTs are dynamic digital representations of physical assets that integrate various technologies such as artificial intelligence, machine learning, and data analytics to simulate real-world conditions and predict future performance. This section reviews the state-of-the-art in DT development and the integration of DevOps practices for the engineering of DTs, mainly focusing on smart buildings and infrastructure.

The application of DTs in the built environment is relatively nascent but rapidly growing. Early implementations in Smart campuses focus on integrating BIM with real-time data from IoT sensors to create comprehensive digital representations of buildings. The Cambridge campus case study, UK, [17] demonstrated how DTs could be leveraged to improve asset maintenance and equipment fault detection and diagnosic (FDD), highlighting the potential for DTs to enhance the operational efficiency of buildings through real-time monitoring and predictive analytics. [18] explored FDD in a similar fashion on Carlton campus, Canada. On Aalto University campus, Finland, [19] explored a detailed implementation of sensor node mesh on a facade while [20] focused on IoT deployment of the DT with a user-centric perspective, allowing contributions from the community.

A critical challenge in developing effective DTs is the integration of data from disparate and heterogeneous sources in data-driven infrastructure. Technologies such as Ontologies, Extract, Transform, Load (ETL), service-oriented architectures, and data lakes have been employed to facilitate this integration [21–25]. However, issues such as data synchronization and quality remain significant hurdles. [26] discuss the trade-offs between synchronization costs and data staleness, emphasizing the need for continuous data streams to maintain the accuracy and reliability of DTs.

The integration of DevOps practices in the development and operation of DTs is a relatively new area of research. DevOps, which emphasizes continuous integration and continuous deployment (CI/CD), automated testing, and monitoring, can significantly enhance the lifecycle management of DTs. Some studies (e.g., [27]) illustrate how DevOps can be adapted for IoT systems, providing fast and continuous feedback on system availability and performance. Applying these principles to DTs can improve their responsiveness and adaptability, ensuring that they remain accurate reflections of their physical counterparts.

A growing body of work, mainly coming from the Model-Based Engineering System community, tries to transfer pure software based DevOps practices to CPS. Those show the ability and potential benefits of applying selected DevOps capabilities in safety-critical embedded systems. [9] They tend to use DT in CPS to enable DevOps [8, 11, 28] as opposed to DevOps for the DT. The concerns revolve around accelerating deployments and run-time telemetry to gain real time data for future design improvements. [7]. [29] follows the same path, but add testing phase to the exploration. Within the large scale AIDOaRt project, [30] bring AI tools into the mix.

Other related technologies and applications of DTs include augmented reality (AR) for enhanced asset visualization and management, as demonstrated by the AR-based maintenance system proposed by the Cambridge team [17]. Additionally, the integration of DTs with cloud platforms allows for scalable and flexible data management and processing [24], further extending their applicability in various domains of asset management and smart building operations [21].

8 CONCLUSION AND FUTURE WORK

In this paper, we presented JuNo-OPS a DevOps framework for the engineering of DTs for built assets. The framework has been developed, tested and validated in the the context of a multi-function room at ETS. It directly addresses the three essential characteristics described in the introduction (DTs are complex software systems; DTs are associated with systems composed of multiple aspects; and the development of DTs is a multi-disciplinary process). The main focus of the paper is on the microservices architecture of the Digital Twin (DT) software and the DevOps infrastructure that is used to support the development, continuous integration, and continuous delivery of the DT software. By automating main phases of the DevOps process, the JuNo-OPS allows domain experts with no software engineering background to benefit from the best practices of DevOps and software engineering and concentrate on the core value that they can contribute to the engineering of DTs. This way fruitful collaborations can be established in such a way that a set of complementary expertise and domain knowledge can contribute to the development of DTs that address a broad range of system aspects.

While the JuNo-OPS has been developed and partly validated in the GRIDD lab, the next validation phase will include the use of the framework in upcoming research projects that will include the development of a DT for a research climatic room and a DT for an experimental floor architecture project that will start next September, both at ETS. These projects will contribute to the incremental evolution of the framework.

Finally, future research should continue to explore the synergies between DTs and DevOps, focusing on overcoming the challenges of data (integration, synchronization, quality management, security and privacy) and people (domain experts and software engineers) to realize the full potential of DTs in the built environment.

REFERENCES

[1] Arup, "Digital twin: Towards a meaningful framework," Arup: London, UK, Tech. Rep., 2019.
[2] J. Mertens, S. Klikovits, F. Bordeleau, J. Denil, and Ø. Haugen, "Continuous evolution of digital twins using the dartwin notation," *Software and Systems Modeling*, Accepted for publication, 2024.
[3] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.
[4] D. Lehner, A. Garmendia, and M. Wimmer, "Towards flexible evolution of digital twins with fluent apis," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 1–4.
[5] J. Robles, C. Martín, and M. Díaz, "Opentwins: An open-source framework for the development of next-gen compositional digital twins," *Computers in Industry*, 2023.
[6] M. K. Traoré, "A conceptual framework to analyze urban digital twins interoperability," 2024.
[7] J. Hugues, A. Hristosov, J. J. Hudak, and J. Yankel, "Twinops-devops meets model-based engineering and digital twins for the engineering of cps," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–5.
[8] M. Ugarte Querejeta, L. Etxeberria, and G. Sagardui, "Towards a devops approach in cyber physical production systems using digital twins," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2020, pp. 205–216.
[9] J. Dobaj, A. Riel, T. Krug, M. Seidl, G. Macher, and M. Egretzberger, "Towards digital twin-enabled devops for cps providing architecture-based service adaptation & verification at runtime," in *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2022, pp. 132–143.
[10] R. Kostromin and A. Feoktistov, "Agent-based devops of software and hardware resources for digital twins of infrastructural objects," in *Proceedings of the 4th International Conference on Future Networks and Distributed Systems*, 2020, pp. 1–6.
[11] J. Mertens and J. Denil, "The digital twin as a common knowledge base in devops to support continuous system evolution," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2021, pp. 158–170.
[12] J. M. Grünwaldt, "Digital twin for devops: Data-driven continuous monitoring of devops systems and teams," Master's thesis, 2023.
[13] J. Mertens and J. Denil, "Digital twins for continuous deployment in model-based systems engineering of cyber-physical systems," in *CEUR workshop proceedings*, vol. 2822, 2020, pp. 32–39.
[14] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution, 2021.
[15] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations*. IT Revolution, 2018.
[16] P. Sharafdin, "Building occupant comfort monitoring through digital twins using plug-and-play iot sensors," Master's thesis, École de technologie supérieur, Université du Québec, 2024.
[17] V. Qiuchen Lu, A. K. Parlikad, P. Woodall, G. D. Ranasinghe, and J. Heaton, "Developing a dynamic digital twin at a building level: Using cambridge campus as case study," in *International Conference on Smart Infrastructure and Construction 2019 (ICSIC) Driving data-informed decision-making*. ICE Publishing, 2019, pp. 67–75.
[18] Z. Shi, A. Abdelalim, W. O'Brien, R. Attar, P. Akiki, K. Graham, and A. Tessier, "Digital Campus Innovation Project: Integration of Building Information Modelling with Building Performance Simulation and Building Diagnostics." [Online]. Available: https://www.research.autodesk.com/app/uploads/2023/03/digital-campus-innovation-project.pdf_recAvB2eHblNK6Jra.pdf
[19] S. H. Khajavi, N. H. Motlagh, A. Jaribion, L. C. Werner, and J. Holmström, "Digital twin: vision, benefits, boundaries, and creation for buildings," *IEEE access*, vol. 7, pp. 147 406–147 419, 2019.
[20] B. Dave, A. Buda, A. Nurminen, and K. Främling, "A framework for integrating BIM and IoT through open standards," vol. 95, pp. 35–45. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580517305964

[21] Q. Lu, A. K. Parlikad, P. Woodall, G. Don Ranasinghe, X. Xie, Z. Liang, E. Konstantinou, J. Heaton, and J. Schooling, "Developing a digital twin at building and city levels: Case study of west cambridge campus," *Journal of Management in Engineering*, vol. 36, no. 3, p. 05020004, 2020.

[22] P. Vassiliadis, "A survey of extract–transform–load technology," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 5, no. 3, pp. 1–27, 2009.

[23] L. Chamari, P. Pauwels, E. Petrova, E. Chochanova, R. Sebastian, N. Mutsaers, B. Veldhuis, S. Hopkins, and p. u. family=Jong, given=Niels, "Study of data needs and requirements in smart buildings." [Online]. Available: https://brains4buildings.org/wp-content/uploads/2022/05/B4B-WP4-D4.03_Data-Needs-and-Requirements-1.pdf

[24] L. Chamari, E. Petrova, and P. Pauwels, "An End-to-End Implementation of a Service-Oriented Architecture for Data-Driven Smart Buildings," vol. 11, pp. 117 261–117 281. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10287934

[25] A. C. Marosi, M. Emődi, A. Farkas, R. Lovas, R. Beregi, G. Pedone, B. Németh, and P. Gáspár, "Toward Reference Architectures: A Cloud-Agnostic Data Analytics Platform Empowering Autonomous Systems," vol. 10, pp. 60 658–60 673. [Online]. Available: https://ieeexplore.ieee.org/document/9789158/?arnumber=9789158

[26] X. S. Qu and Z. Jiang, "A time-based dynamic synchronization policy for consolidated database systems," *MIS Quarterly, Forthcoming*, 2018.

[27] M. A. López-Peña, J. Díaz, J. E. Pérez, and H. Humanes, "Devops for iot systems: Fast and continuous monitoring feedback of system availability," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 695–10 707, 2020.

[28] J. Mertens and J. Denil, "Digital twins for continuous deployment in model-based systems engineering of cyber-physical systems," in *CEUR Workshop Proceedings*, pp. p. 32–39. [Online]. Available: http://ceur-ws.org/Vol-2822/p4.pdf

[29] B. Combemale and M. Wimmer, "Towards a Model-Based DevOps for Cyber-Physical Systems," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds. Springer International Publishing, pp. 84–94.

[30] R. Eramo, V. Muttillo, L. Berardinelli, H. Bruneliere, A. Gomez, A. Bagnato, A. Sadovykh, and A. Cicchetti, "AIDOaRt: AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 303–310. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9556443

LISTE DE RÉFÉRENCES

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J. & Clarke, P. (2023). Decomposition of monolith applications into microservices architectures : A systematic review. *IEEE Transactions on Software Engineering*, 49(8), 4213–4242.

Ahmadi, H., Nag, A., Khar, Z., Sayrafian, K. & Rahardja, S. (2021). Networked twins and twins of networks : An overview on the relationship between digital twins and 6G. *IEEE Communications Standards Magazine*, 5(4), 154–160.

Aissat, S., Beaulieu, J., Bordeleau, F., Motamedi, A. & Poirier, É. A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins. *cit. on*, 5.

Aissat, S., Beaulieu, J., Bordeleau, F., Gascon-Samson, J., Poirier, E. A. & Motamedi, A. (2024a). JuNo-OPS : A DevOps Framework for the Engineering of Digital Twins for Built Assets. *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pp. 496–506.

Aissat, S., Beaulieu, J., Bordeleau, F., Motamedi, A. & Poirier, E. A. (2024b). A DevOps Approach for the Systematic Development and Evolution of Built Assets Digital Twins. *Proceedings of the CIB W78 Conference 2024, October 1st-3rd 2024, Marrakesh, Morocco*.

Akanmu, A. A., Anumba, C. J. & Ogunseiju, O. O. (2021). Towards next generation cyber-physical systems and digital twins for construction. *Journal of Information Technology in Construction*, 26.

Anderson, D. J. (2010). *Kanban : successful evolutionary change for your technology business*. Blue hole press.

Arup. (2019). *Digital Twin : Towards a Meaningful Framework*.

Bass, L., Weber, I. & Zhu, L. (2015). *DevOps : A software architect's perspective*. Addison-Wesley Professional.

Bendraou, R., Combemale, B., Crégut, X. & Gervais, M.-P. (2007). Definition of an Executable SPEM 2.0. *14th Asia-Pacific Software Engineering Conference (APSEC'07)*, pp. 390–397.

Braun, A., Tuttas, S., Stilla, U. & Borrmann, A. (2018). BIM-Based Progress Monitoring. Dans Borrmann, A., König, M., Koch, C. & Beetz, J. (Éds.), *Building Information Modeling : Technology Foundations and Industry Practice* (pp. 463–476). München : Springer International Publishing. Repéré à https://doi.org/10.1007/978-3-319-92862-3_28.

Brilakis, I., Pan, Y., Borrmann, A., Mayer, H.-G., Rhein, F., Vos, C., Pettinato, E. & Wagner, S. (2019). "Built Environment Digital Twining". *International Workshop on Built Environment Digital Twinning*. doi : 10.17863/CAM.65445.

Chamari, L., Pauwels, P., Petrova, E., Dubbeldam, J.-W. & Jong, N. D. *Deliverable n°D-4.0.6*. Reference Architecture for Smart Buildings. Brains for Building's Energy Systems. Repéré à https://brains4buildings.org/wp-content/uploads/2023/08/B4B-WP4-D4.06_Reference_Architecture-FINAL.pdf.

Chamari, L., Pauwels, P., Petrova, E., Dubbeldam, J., de Jong, N. & Gunderi, K. M. (2023a). Reference Architecture for Smart Buildings.

Chamari, L., Petrova, E. & Pauwels, P. (2023b). An end-to-end implementation of a service-oriented architecture for data-driven smart buildings. *Ieee Access*, 11, 117261–117281.

Chen, Y.-C. & Hsieh, T.-C. (2014). Big data for digital government : Opportunities, challenges, and strategies. *International journal of public administration in the digital age (IJPADA)*, 1(1), 1–14.

Chenari, B., Carrilho, J. D. & Da Silva, M. G. (2016). Towards sustainable, energy-efficient and healthy ventilation strategies in buildings : A review. *Renewable and Sustainable Energy Reviews*, 59, 1426–1447.

Combemale, B. & Wimmer, M. Towards a Model-Based DevOps for Cyber-Physical Systems. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pp. 84–94. doi : 10.1007/978-3-030-39306-9_6.

Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4), 28–31.

Davari, S., Shahinmoghadam, M., Motamedi, A. & Poirier, E. (2022). Demystifying the definition of digital twin for built environment. *International conference on construction engineering and project management*, pp. 1122–1129.

Dave, B., Buda, A., Nurminen, A. & Främling, K. A Framework for Integrating BIM and IoT through Open Standards. 95, 35–45. doi : 10.1016/j.autcon.2018.07.022.

Dihan, M. S., Akash, A. I., Tasneem, Z., Das, P., Das, S. K., Islam, M. R., Islam, M. M., Badal, F. R., Ali, M. F., Ahamed, M. H. et al. (2024). Digital twin : Data exploration, architecture, implementation and future. *Heliyon*, 10(5).

Djebali, S., Guerard, G. & Taleb, I. Survey and Insights on Digital Twins Design and Smart Grid's Applications. 153, 234–248. doi : 10.1016/j.future.2023.11.033.

Dobaj, J., Riel, A., Krug, T., Seidl, M., Macher, G. & Egretzberger, M. (2022). Towards digital twin-enabled DevOps for CPS providing architecture-based service adaptation & verification at runtime. *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 132–143.

Eramo, R., Muttillo, V., Berardinelli, L., Bruneliere, H., Gomez, A., Bagnato, A., Sadovykh, A. & Cicchetti, A. AIDOaRt : AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems. *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 303–310. doi : 10.1109/DSD53832.2021.00053.

Eramo, R., Muttillo, V., Berardinelli, L., Bruneliere, H., Gomez, A., Bagnato, A., Sadovykh, A. & Cicchetti, A. (2021). Aidoart : Ai-augmented automation for devops, a model-based framework for continuous development in cyber-physical systems. *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 303–310.

Fanger, P. O. (1970). Thermal comfort. Analysis and applications in environmental engineering.

Forsgren, N., Humble, J. & Kim, G. (2018). *Accelerate : The science of lean software and devops : Building and scaling high performing technology organizations*. IT Revolution.

Forsgren, N., Smith, D., Humble, J. & Frazelle, J. (2019). 2019 accelerate state of devops report. *DORA and Google Cloud, Tech. Rep*, 48455.

Fritzsch, J., Bogner, J., Haug, M., Franco da Silva, A. C., Rubner, C., Saft, M., Sauer, H. & Wagner, S. (2023). Adopting microservices and DevOps in the cyber-physical systems domain : A rapid review and case study. *Software : Practice and Experience*, 53(3), 790–810.

GitLab. (2022). What are the benefits of a microservices architecture ? Repéré à https://about.gitlab.com/fr-fr/blog/2022/09/29/what-are-the-benefits-of-a-microservices-architecture/.

Glaessgen, E. & Stargel, D. (2012). The digital twin paradigm for future NASA and US Air Force vehicles. *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, pp. 1818.

Greif, T., Stein, N. & Flath, C. M. (2020). Peeking into the void : Digital twins for construction site logistics. *Computers in Industry*, 121, 103264. doi : 10.1016/j.compind.2020.103264.

Grieves, M. W. (2023). Digital twins : past, present, and future. Dans *The digital twin* (pp. 97–121). Springer.

Hevner, A. & Chatterjee, S. (2010). *Design Research in Information Systems : Theory and Practice*. Boston, MA : Springer US. doi : 10.1007/978-1-4419-5653-8.

Hosamo, H. H., Imran, A., Cardenas-Cartagena, J., Svennevig, P. R., Svidt, K. & Nielsen, H. K. A Review of the Digital Twin Technology in the AEC-FM Industry. 2022(1), 2185170. doi : 10.1155/2022/2185170.

Hugues, J., Hristosov, A., Hudak, J. J. & Yankel, J. TwinOps - DevOps Meets Model-Based Engineering and Digital Twins for the Engineering of CPS. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems : Companion Proceedings*, (MODELS '20), 1–5. doi : 10.1145/3417990.3421446.

Humble, J. & Farley, D. (2010). Continuous delivery : Reliable software releases through build. *Test, and deployment automation. Pearson Education*, 1.

Ikemoto, G. S. & Marsh, J. A. (2007). Cutting through the "data-driven" mantra : Different conceptions of data-driven decision making. *Teachers College Record*, 109(13), 105–131.

Jones, D., Snider, C., Nassehi, A., Yon, J. & Hicks, B. (2020). Characterising the Digital Twin : A systematic literature review. *CIRP journal of manufacturing science and technology*, 29, 36–52.

Jones, P., Clarke-Hill, C., Shears, P., Comfort, D. & Hillier, D. (2004). Radio frequency identification in the UK : opportunities and challenges. *International Journal of Retail & Distribution Management*, 32(3), 164–171.

Kamel Boulos, M. N. & Zhang, P. (2021). Digital twins : from personalised medicine to precision public health. *Journal of personalized medicine*, 11(8), 745.

Kang, S.-O., Lee, E.-B. & Baek, H.-K. (2019). A digitization and conversion tool for imaged drawings to intelligent piping and instrumentation diagrams (P&ID). *Energies*, 12(13), 2593.

Karamat, O., Lahlali, N. & Mercier, P. (2023). *Développement d'un jumeau numérique (Digital Twin) pour les bâtiments intelligents*. Repéré à https://gridd.etsmtl.ca/.

Khajavi, S. H., Motlagh, N. H., Jaribion, A., Werner, L. C. & Holmström, J. (2019). Digital twin : vision, benefits, boundaries, and creation for buildings. *IEEE access*, 7, 147406–147419.

Khan, L. U., Han, Z., Saad, W., Hossain, E., Guizani, M. & Hong, C. S. (2022). Digital twin of wireless systems : Overview, taxonomy, challenges, and opportunities. *IEEE Communications Surveys & Tutorials*, 24(4), 2230–2254.

Kim, G., Humble, J., Debois, P., Willis, J. & Forsgren, N. (2021). *The DevOps handbook : How to create world-class agility, reliability, & security in technology organizations*. It Revolution.

Kostromin, R. & Feoktistov, A. Agent-Based DevOps of Software and Hardware Resources for Digital Twins of Infrastructural Objects. *Proceedings of the 4th International Conference on Future Networks and Distributed Systems*, (ICFNDS '20), 1–6. doi : 10.1145/3440749.3442599.

Kritzinger, W., Karner, M., Traar, G., Henjes, J. & Sihn, W. (2018). Digital Twin in manufacturing : A categorical literature review and classification. *Ifac-PapersOnline*, 51(11), 1016–1022.

Li, L., Aslam, S., Wileman, A. & Perinpanayagam, S. (2021). Digital twin in aerospace industry : A gentle introduction. *IEEE Access*, 10, 9543–9562.

Li, X., Feng, M., Ran, Y., Su, Y., Liu, F., Huang, C., Shen, H., Xiao, Q., Su, J., Yuan, S. et al. (2023). Big Data in Earth system science and progress towards a digital twin. *Nature Reviews Earth & Environment*, 4(5), 319–332.

Lom, M., Pribyl, O. & Svitek, M. (2016). Industry 4.0 as a part of smart cities. *2016 Smart Cities Symposium Prague (SCSP)*, pp. 1–6.

López-Peña, M. A., Díaz, J., Pérez, J. E. & Humanes, H. (2020). DevOps for IoT systems : Fast and continuous monitoring feedback of system availability. *IEEE Internet of Things Journal*, 7(10), 10695–10707.

Lu, Q., Parlikad, A. K., Woodall, P., Don Ranasinghe, G., Xie, X., Liang, Z., Konstantinou, E., Heaton, J. & Schooling, J. (2020). Developing a digital twin at building and city levels : Case study of West Cambridge campus. *Journal of Management in Engineering*, 36(3), 05020004.

Madubuike, O. C., Anumba, C. J. & Khallaf, R. (2022). A review of digital twin applications in construction. *Journal of Information Technology in Construction (ITcon)*, 27(8), 145–172. doi : 10.36680/j.itcon.2022.008.

Manifesto, A. (2001). Manifesto for agile software development.

Marosi, A. C., Emodi, M., Hajnal, , Lovas, R., Kiss, T., Poser, V., Antony, J., Bergweiler, S., Hamzeh, H., Deslauriers, J. & Kovács, J. Interoperable Data Analytics Reference Architectures Empowering Digital-Twin-Aided Manufacturing. 14(4), 114. doi : 10.3390/fi14040114.

Marosi, A. C., Emődi, M., Farkas, A., Lovas, R., Beregi, R., Pedone, G., Németh, B. & Gáspár, P. Toward Reference Architectures : A Cloud-Agnostic Data Analytics Platform Empowering Autonomous Systems. 10, 60658–60673. doi : 10.1109/ACCESS.2022.3180365.

Meijer, C., Uh, H.-W. & El Bouhaddani, S. (2023). Digital twins in healthcare : Methodological challenges and opportunities. *Journal of Personalized Medicine*, 13(10), 1522.

Merabet, G. H., Essaaidi, M., Haddou, M. B., Qolomany, B., Qadir, J., Anan, M., Al-Fuqaha, A., Abid, M. R. & Benhaddou, D. (2021). Intelligent building control systems for thermal comfort and energy-efficiency : A systematic review of artificial intelligence-assisted techniques. *Renewable and Sustainable Energy Reviews*, 144, 110969.

Mertens, J. & Denil, J. Digital Twins for Continuous Deployment in Model-Based Systems Engineering of Cyber-Physical Systems. *CEUR Workshop Proceedings*, pp. p. 32-39. Repéré à http://ceur-ws.org/Vol-2822/p4.pdf.

Mertens, J. & Denil, J. (2021). The digital twin as a common knowledge base in devops to support continuous system evolution. *International Conference on Computer Safety, Reliability, and Security*, pp. 158–170.

Mertens, J., Klikovits, S., Bordeleau, F., Denil, J. & Haugen, Ø. (2024). Continuous evolution of digital twins using the dartwin notation. *Software and Systems Modeling*.

Mertens, J., Klikovits, S., Bordeleau, F., Denil, J. & Haugen, Ø. (Accepted for publication, 2024). Continuous Evolution of Digital Twins using the DarTwin Notation. *Software and Systems Modeling*.

Moeller, D. P. (2003). *Mathematical and computational modeling and simulation*. Springer.

Mylonas, G., Kalogeras, A., Kalogeras, G., Anagnostopoulos, C., Alexakos, C. & Muñoz, L. (2021). Digital twins from smart manufacturing to smart cities : A survey. *Ieee Access*, 9, 143222–143249.

Nunna, S., Kousaridas, A., Ibrahim, M., Dillinger, M., Thuemmler, C., Feussner, H. & Schneider, A. (2015). Enabling real-time context-aware collaboration through 5G and mobile edge computing. *2015 12th International Conference on Information Technology-New Generations*, pp. 601–605.

of Heating, A. S., Refrigerating, Engineers, A.-C. & Institute, A. N. S. (2004). *Thermal environmental conditions for human occupancy*. American Society of Heating, Refrigerating and Air-Conditioning Engineers.

Ozkaya, I., Kruchten, P., Nord, R. L. & Brown, N. (2011). Managing technical debt in software development : report on the 2nd international workshop on managing technical debt, held at ICSE 2011. *ACM SIGSOFT Software Engineering Notes*, 36(5), 33–35.

Pires, F., Cachada, A., Barbosa, J., Moreira, A. P. & Leitão, P. (2019). Digital twin in industry 4.0 : Technologies, applications and challenges. *2019 IEEE 17th international conference on industrial informatics (INDIN)*, 1, 721–726.

Piromalis, D. & Kantaros, A. (2022). Digital twins in the automotive industry : The road toward physical-digital convergence. *Applied System Innovation*, 5(4), 65.

Purcell, W. & Neubauer, T. (2023). Digital Twins in Agriculture : A State-of-the-art review. *Smart Agricultural Technology*, 3, 100094.

Sanchez-Iborra, R., Sanchez-Gomez, J. & Skarmeta, A. (2018). Evolving IoT networks by the confluence of MEC and LP-WAN paradigms. *Future Generation Computer Systems*, 88, 199–208.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1), 30–39.

Scherma, V. (2024). *Design and implementation of an integrated DevOps framework for Digital Twins as a Service software platform*. (Thèse de doctorat, Politecnico di Torino).

Shahzad, M., Shafiq, M. T., Douglas, D. & Kassem, M. (2022). Digital twins in built environments : an investigation of the characteristics, applications, and challenges. *Buildings*, 12(2), 120.

Sharafdin, P. (2024). *Building occupant comfort monitoring through Digital Twins using plug-and-play IoT sensors*. (Mémoire de maîtrise, École de technologie supérieur, Université du Québec).

Sharma, A., Kosasih, E., Zhang, J., Brintrup, A. & Calinescu, A. (2022). Digital Twins : State of the art theory and practice, challenges, and open research questions. *Journal of Industrial Information Integration*, 30, 100383.

Shi, Z., Abdelalim, A., O'Brien, W., Attar, R., Akiki, P., Graham, K. & Tessier, A. Digital Campus Innovation Project : Integration of Building Information Modelling with Building Performance Simulation and Building Diagnostics. Repéré à https://www.research.autodesk.com/app/uploads/2023/03/digital-campus-innovation-project.pdf_recAvB2eHblNK6Jra.pdf.

Si, L. & Baier, H. (2015). Real-time impact visualization inspection of aerospace composite structures with distributed sensors. *Sensors*, 15(7), 16536–16556.

Singh, M., Fuenmayor, E., Hinchy, E. P., Qiao, Y., Murray, N. & Devine, D. (2021). Digital twin : Origin to future. *Applied System Innovation*, 4(2), 36.

Talasila, P., Gomes, C., Mikkelsen, P. H., Arboleda, S. G., Kamburjan, E. & Larsen, P. G. (2023). Digital twin as a service (DTaaS) : a platform for digital twin developers and users. *2023 IEEE Smart World Congress (SWC)*, pp. 1–8.

Tao, F., Zhang, H., Liu, A. & Nee, A. Y. (2018). Digital twin in industry : State-of-the-art. *IEEE Transactions on industrial informatics*, 15(4), 2405–2415.

Tien, J. M. (2017). Internet of things, real-time decision making, and artificial intelligence. *Annals of Data Science*, 4, 149–178.

Tisi, M., Bruneliere, H., de Lara, J., Di Ruscio, D. & Kolovos, D. (2021). Towards Twin-Driven Engineering : Overview of the State-of-The-Art and Research Directions. *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*, pp. 351–359. doi : 10.1007/978-3-030-85874-2_37.

Trabucchi, D., Buganza, T. & Pellizzoni, E. (2017). Give Away Your Digital Services : Leveraging Big Data to Capture Value New models that capture the value embedded in the data generated by digital services may make it viable for companies to offer those services for free. *Research-Technology Management*, 60(2), 43–52.

Ugarte Querejeta, M., Etxeberria, L. & Sagardui, G. Towards a DevOps Approach in Cyber Physical Production Systems Using Digital Twins. *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, pp. 205–216. doi : 10.1007/978-3-030-55583-2_15.

Van der Valk, H., Haße, H., Möller, F., Arbter, M., Henning, J.-L. & Otto, B. (2020). A Taxonomy of Digital Twins. *AMCIS*.

Yang, B., Lv, Z. & Wang, F. (2022). Digital twins for intelligent green buildings. *Buildings*, 12(6), 856.

Yaqoob, I., Khan, L. U., Kazmi, S. A., Imran, M., Guizani, N. & Hong, C. S. (2019). Autonomous driving cars in smart cities : Recent advances, requirements, and challenges. *IEEE Network*, 34(1), 174–181.

Zhao, J., Feng, H., Chen, Q. & Garcia de Soto, B. (2022). Developing a conceptual framework for the application of digital twin technologies to revamp building operation and maintenance processes. *Journal of Building Engineering*, 49. doi : 10.1016/j.jobe.2022.104028.

Zheng, X., Lu, J. & Kiritsis, D. (2022). The emergence of cognitive digital twin : vision, challenges and opportunities. *International Journal of Production Research*, 60(24), 7610–7632.