
[All ETDs from UAB](#)

[UAB Theses & Dissertations](#)

2023

A Simulation Framework for Identity and Access Management Based on Internet of Things Architecture

Vahid Moghaddasi
University Of Alabama At Birmingham

Follow this and additional works at: <https://digitalcommons.library.uab.edu/etd-collection>



Part of the [Engineering Commons](#)

Recommended Citation

Moghaddasi, Vahid, "A Simulation Framework for Identity and Access Management Based on Internet of Things Architecture" (2023). *All ETDs from UAB*. 393.
<https://digitalcommons.library.uab.edu/etd-collection/393>

This content has been accepted for inclusion by an authorized administrator of the UAB Digital Commons, and is provided as a free open access item. All inquiries regarding this item or the UAB Digital Commons should be directed to the [UAB Libraries Office of Scholarly Communication](#).

A SIMULATION FRAMEWORK FOR IDENTITY AND ACCESS MANAGEMENT
BASED ON INTERNET OF THINGS ARCHITECTURE

by

VAHID MOGHADDASI

LEON JOLOLIAN, COMMITTEE CHAIR
KARTIKEYAN LINGASUBRAMANIAN
ABDOLLAH S. MIRBOZORGI
MURAT M. TANIK
B. EARL WELLS

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2023

ProQuest Number: XXXXX

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest xxxxxxxxxxxxxx

Published by ProQuest LLC (2023). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A PROCESS FRAMEWORK FOR IDENTITY AND ACCESS MANAGEMENT BASED ON INTERNET OF THINGS ARCHITECTURE

VAHID MOGHADDASI

COMPUTER ENGINEERING

ABSTRACT

The swift progression of the Internet of Things (IoT) architecture has revolutionized various industrial processes, including enterprise Identity and Access Management (IAM). This dissertation develops a dynamic process framework for Identity and Access Management based on the Internet of Things architecture. System analysis with Digital Twin simulation explores the integration of IoT devices to enhance IAM in enterprises. Digital Twin is a virtual service that enabled us the creation of a comprehensive digital IAM environment to explore alternative designs.

This dissertation begins by providing an overview of IoT architecture and its impact on enterprise IAM. It discusses the challenges enterprises face in managing identities and access in an increasingly connected world. The dissertation then introduces Digital Twin and its role in IoT, explaining how it can create a virtual representation of IAM processes and their lifecycle.

The focus of this dissertation is on creating a process framework for IAM-based IoT architecture and Digital Twin simulation. This dissertation presents a detailed analysis of how Digital Twin can simulate IAM scenarios, test IAM policies, and predict the

impact of IAM changes in a controlled environment. This can lead to improved security, efficiency, and compliance.

Keywords: Internet of Things architecture, Enterprise Systems, Process Improvement, Identity and Access Management, digital twin, Drag-and-Drop, IoT, Digital Twin

DEDICATION

I would like to dedicate my dissertation to my wife, daughter, siblings, and friends for their support and encouragement.

ACKNOWLEDGMENTS

I want to thank my advisor Dr. Leon Jololian, who has given me the opportunity to start my PhD after many years of industry work. I want to thank Dr. Karthik Lingasubramanian for believing in me early on and accepting me as his PhD student. I want to thank my committee members for serving on my advisory committee and providing feedback on my progress.

I would like to thank Dr. Murat M. Tanik for introducing me to Dr. Raymond T. Yeh, who shared his insight on Process Improvement and experience with me. I would like to thank Dr. Murat M. Tanik and Dr. Abdollah S. Mirbozorgi for their corrective recommendations and insight. I would like to thank my manager Lewis Liu for facilitating financially the PhD program. Last, I would like to thank my friend Dave Wong for his support and help.

TABLE OF CONTENTS

	<i>Page</i>
ABSTRACT	iv
DEDICATION	vi
ACKNOWLEDGMENTS	vii
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
 CHAPTER	
1 INTRODUCTION	1
Overview and Dissertation Content	2
Summary	5
2 LITERATURE REVIEW	6
Identity and Access Management as a Process.	7
Report Access Model Sub-process of IAM	8
Report Workflow Sub-process	10
Identity and Access Management Process using Azure Digital Twins	12
Design of IoT Device and Models for Identity and Access Management.....	12
Digital Twin in Design and Implementation of IAM Process	15
Azure technology in Design and Implementation of IAM process	16
Design And Implementation of IAM Process	19
Enterprise Systems and Enterprise Architecture	20
Process Improvement Using Quality Definitions	21
Achieving Quality Improved Process	22
A Context for Process Improvement	23
Process Improvement Activities and Modeling	24
Components of Enterprise Design and Lifecycle	26
Engineering of Enterprises	28
A Process-Centered Engineering Quality Model	30

A General Systems Design Methodology	31
Edges-in Design	33
Multileveled Systems Design	35
Requirements Analysis and System Specification	39
Requirements Analysis	40
3 SYSTEM DESIGN AS A STRUCTURED PROCESS	45
A General System Perspective	46
Structural Attributes of Design Methodologies	47
Internet of Things Sensors Architecture	48
Structural Attributes	49
Design Strategies	51
The Significant-Components Approach	54
Constraint-Driven Design	54
Control-Driven Approaches	56
Design Mechanisms	57
Starting Point, Scope, and Scale of Design Methodologies	57
Complexity and Integrity	60
4 INCORPORATING DIGITAL TWIN TECHNOLOGY INTO THE FRAMEWORK: CASE STUDY	62
Current Identity and Access Management Issues	62
Requirements and Constraints	63
Quantifiable Constraints	64
Nonquantifiable Constraints	65
Logical Design	66
Partitioning of System Functions.....	67
Logical Architectures and Virtual Machines	71
Partitioning and Allocation to Hardware Sensor Systems	73
Partitioning	75
5 DIGITAL TWIN SIMULATION WITH CONSTRAINTS	77
Digital Twin and Identity and Access Management	78
Implementation Details using Azure	78
Creating an Azure Digital Twins Instance	79
Creating Resources	80
Creating The Digital Twin Models	82
Validating the DTDL Models	84
Azure Digital Twins Explorer	85
Importing Data from Spreadsheet	88
Importing Data from JavaScript Object Notation File	89
Use The Command Line Interface to Create New Digital Twin	90
Establishing Relationship Between IoT Digital Twin Models	91

Creating an IoT Hub	92
Adding Devices to Internet of Things Hub	94
Creating an Azure Function App	97
Building Function to Ingest the Telemetry Data	99
Event Subscription in Dealing with IAM Exceptions	100
Developing Simulator to Send Telemetry Data to IoT Hub	100
End To End Simulation for IoT Functionality	102
6 SUMMARY, CONCLUSIONS, AND FUTURE WORK	104
LIST OF REFERENCES	108

LIST OF TABLES

<i>Table</i>	<i>Page</i>
1 List of control objectives in IAM process	14
2 Challenges to Identity and Access Management	18
3 Spectrum of product quality attribute emphasis	22
4 Spectrum of engineering design emphasis	31
5 Current Issues	43
6 Deficiencies and current approaches.....	43
7 Addressing the needs.....	44
8 CSV format of DTMI for bulk load into Digital Twin	89
9 Case Study Scenario	103
10 Summary of applying the end-to-end Simulation	106
11 Summary and Conclusion	107

LIST OF FIGURES

<i>Figure</i>	<i>Page</i>
1 A Dynamic Identity and Access Management Process (DYN-IAMP) framework ..	1
2 IAM as a Process	8
3 Report Access Model Subprocess	10
4 Subprocess of Report Workflow	11
5 Report Identity Subprocess	12
6 Access Modeling Process	15
7 Enterprise system	19
8 Process-centered improvement operational concept	23
9 Process-Centered engineering quality model	24
10 Relationship between enterprise development and operation	28
11 The three layers of enterprise architecture	29
12 Locating Azure Digital Twins in Azure Marketplace	79
13 Creating Azure Digital Twins resource	80
14 Azure Access control	81
15 List and assign roles in Azure Access control panel	82
16 JSON code to ingest data from sensors into devices in Azure IoT	83
17 Relationships between Digital Twin models	84
18 Compiling DTDL Validator using dotnet utility	85
19 Successful validation of models	85

20 Retrieving Azure Digital Twins instance URL	86
21 Entering Azure Digital Twins URL in Explorer	87
22 Azure Digital Twins Explorer showing the twin	88
23 JSON code to upload bulk data into ADT.....	90
24 Azure Digital Twins CLI	91
25 Model graph with established relationship	92
26 Creating IoT hub and being deployed to Azure cloud	93
27 IoT hub is created and deployed to Azure cloud	93
28 Creating provisioning IoT device	95
29 Creating authentication IoT device	95
30 List of all devices that we created in IoT hub	96
31 Device ID, Shared key, and connection string	97
32 List all Function App.....	98
33 Function App configuration	99
34 Building the simulator	101
35 Telemetry Simulator	102
36 IAM Summary	104
37 Internet of Things end-to-end Simulation	106

LIST OF ABBREVIATIONS

ADT	Azure Digital Twins
AI	Artificial Intelligence
CLI	Command Line Interface
CSV	Comma-Separated Values
DB	Database
DNS	Domain Name System
DTDL	Digital Twin Definition Language
DTMI	Digital Twin Model Identifier
ESE	Enterprise System Engineering
IAM	Identity and Access Management
IAMP	Identity and Access Management Process
ID	Identification
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
MSAL	Microsoft Authentication Library
SDK	Software Development Kit
SoD	Segregation of Duties
SoR	System of Records
URL	Uniform Resource Locator

CHAPTER 1

INTRODUCTION

In this dissertation, we present a dynamic framework for Identity and Access Management (IAM) [1] based on the Internet of Things architecture analysis with simulation [2]. The goal of our approach is to its responsiveness and continuous use of feedback coming from process monitoring sensors and systems. Simulation is facilitated with digital twin technology [3]. Figure 1 depicts our Dynamic IAM Process framework (DYN-IAMP), which will be explained as we advance in this dissertation.

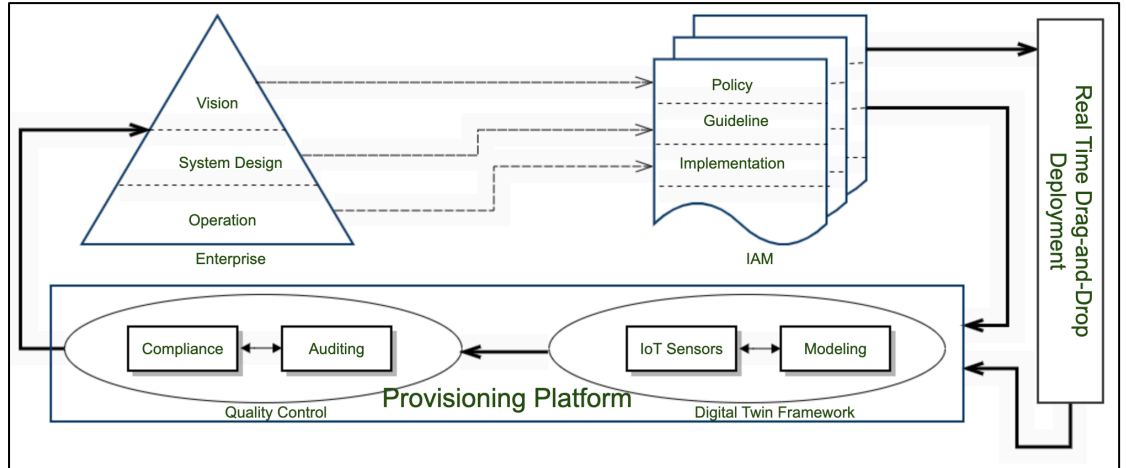


Figure 1. A Dynamic Identity and Access Management Process (DYN-IAMP) framework.

We selected one subsystem to create a pilot process of a large and complex IAM process. The dissertation focuses on the use of Azure Digital Twins simulation for IAM. It

provides a thorough overview of how digital twin can test IAM rules, simulate IAM scenarios, and forecast the effects of IAM changes in a safe setting. Improved security, effectiveness, and compliance in IAM can result from this [2].

Overview and Dissertation Content

In this dissertation, Chapter 2 presents Identity and Access Management as a Process. IAM sub-processes are explored, and the “Report Access Model” sub-process is elaborated. In addition “Report Workflow” sub-process is explained. Digital Twin technology and Digital Twin representation in Azure were also explored. This chapter further addresses the design and implementation of the IAM process. It includes control objectives in the IAM process and Digital Twin in the design process. Azure implementation is also discussed.

Enterprise System Engineering (ESE) is a critical discipline that focuses on the holistic design and management of complex systems within an enterprise. Achieving quality with process modeling is a key aspect of ESE, as it allows for the visualization and analysis of business processes, thereby identifying potential areas for improvement. Quality definition plays a pivotal role in process improvement, as it provides a benchmark against which the effectiveness of processes can be measured. Process improvement activities and modeling are integral to ESE, as they facilitate the systematic identification, analysis, and improvement of existing processes. These activities are often driven by the need to enhance efficiency, reduce waste, and improve overall performance.

The components of enterprise design and lifecycle are also crucial in ESE. These components, which may include system architecture, technology infrastructure, and operational processes, must be carefully designed and managed throughout their lifecycle to

ensure they Persist in addressing the changing requirements of the business sector. The need for enterprise engineering is underscored by the increasing complexity of business environments and the growing interdependence of business processes. A well-designed enterprise system can enhance operational efficiency, improve adaptability, and provide a competitive advantage in the marketplace.

Lastly, a process-centered engineering quality model is essential in ESE. This model focuses on the quality of processes. This way, enterprises can achieve better outcomes, enhance customer satisfaction, and drive business success.

Chapter 4 is on System Design as a Structured Process. This chapter encompasses various methodologies and strategies, each with unique structural attributes. A general system perspective is essential, especially in the Internet of Things sensor architecture, where the interplay of various components is crucial. Design strategies can range from top-down and bottom-up approaches, which focus on the system as a whole or individual component, respectively, to more hybrid methods like the outside-in/inside-out and the most-critical-components-first procedures. Constraint-driven design and control-driven approaches further add to the diversity of design alternatives.

The starting point and scope, and scale of design methodologies can significantly influence the design process. For instance, the edge-in design methodology begins with the system's boundaries. Requirements analysis is a critical step in all design philosophies, helping to identify quantifiable constraints that guide the design process. In the end, these varied design philosophies strive to develop a logical design that satisfies both the system's requirements and constraints, thereby guaranteeing its functionality and efficiency.

Chapter 5 is the case study, Incorporating Digital Twin Technology Into The Framework. This chapter addresses current IAM issues by creating a virtual representation of IAM processes. Implementing this involves creating an Azure Digital Twins instance and generating the necessary resources. Digital Twin models are then created, which can be explored using the Azure Digital Twins Explorer [28]. Data can be imported from spreadsheets or JavaScript Object Notation files, establishing relationships between IoT Digital Twin models. An IoT Hub is created, and devices are added to the Azure Internet of Things Hub. An Azure Function App is then developed, which includes building a function to ingest the telemetry data. An event subscription for the Function App is created, and a simulator is used to send telemetry data to the IoT Hub. This integration of Digital Twin technology can significantly enhance the efficiency and security of IAM processes. At the end of the chapter, we test the End To End operation.

Chapter 6 is on Summary, Conclusions, and Future work. IAM is a crucial yet often overlooked aspect of enterprise security. Unauthorized access and manipulation of private data pose significant challenges. Both government regulations and enterprise policies aim to prevent such unauthorized activities. This dissertation proposes treating IAM as a formal process for provisioning access to digital assets and resources. The complexity of IAM arises from the rapidly changing business environment, necessitating constant monitoring and updating of asset entitlements.

The dissertation presents a dynamic IAM framework based on Internet of Things architecture analysis using digital twin technology. The proposed approach's goal is its responsiveness and continuous use of data feedback from process monitoring sensors and systems. The author's experience with an adaptive business improvement methodology

helped the implementation process. Future potential developments in utilizing artificial intelligence and machine learning are discussed.

Summary

This dissertation introduces a dynamic framework for Identity and Access Management (IAM) based on the analysis of Internet of Things (IoT) architecture using simulation. The framework aims to be responsive and continuously utilize feedback from process monitoring sensors and systems. The simulation aspect is enabled through the use of digital twin technology.

In this dissertation, we identified and addressed two open research questions. These questions are, 1) How to develop a simulation framework for representing the dynamic behavior for access management in a distributed computing environment in an enterprise. 2) How to implement the IoT architecture model for depicting and dealing with real-time scenarios regulating access to distributed resources.

As for future work, there are potential avenues for improvement and advancement in IAM. One area of exploration is the utilization of artificial intelligence and machine learning techniques to enhance IAM processes. These technologies can contribute to more robust and intelligent access control mechanisms. Additionally, ongoing research and development can focus on addressing emerging threats and vulnerabilities in IAM systems, as well as refining the integration of IAM with other cybersecurity measures. Continuous evaluation and adaptation of IAM strategies are essential to keep up with the evolving security landscape and ensure adequate protection of digital assets.

CHAPTER 2

LITERATURE REVIEW

Identity and Access Management (IAM) is a salient part of any mid-size to a large company, which has been long neglected to improve. Access to private data and manipulating that data by unauthorized users has always been a problem and challenge. It is government regulation, as well as the company policy, to prevent any data leak, which costs companies of a large size. In the context of an IAM system, the user lifecycle has external inputs and observable outputs. Inputs are business events, hiring an employee or contractor, changing departments, ending a contract, etc. Outputs are the changes made to integrated systems and applications, creating a login ID, deactivating a login ID, assigning or revoking a group membership, etc [1].

The start of every user's journey involves an onboarding event, which could be a new hire for employees or contractors, sign-up for customers and partners, enrollment for students, and so on. This event leads to the creation of one or more login accounts, along with home directories or mail folders. As time passes, users may change their passwords regularly, and sometimes they may forget or lock themselves out, requiring IT support for password reset or simple lockout. As users transition within the organization, such as changing job functions, projects, or locations, there is often a need to manage and modify their entitlements and other related objects [4].

Every user will inevitably depart at some point, and it's crucial that their access privileges are revoked. Often, user-associated items like home directories, email folders,

and so on, may need to be preserved for a certain period before they can be permanently removed. Certain user information, such as their unique IDs and rehire eligibility, might be kept indefinitely for future use. Some of the most demanding tasks in Identity and Access Management (IAM) processes are outlined below. Tasks such as recognizing additions, modifications, and removals within a system of record (SoR, like HR) and executing corresponding actions - establishing accounts, granting/revoking access, etc., on linked systems and applications are automated. Users have the ability to alter their profiles through self-service requests, such as updating their home phone numbers, and can request new entitlements, like access to a specific application or folder. Through delegated administration, managers, application owners, and other relevant parties can request modifications to identities and privileges within their purview [4]. Periodically, individuals in charge of applications or data must review the users and their security privileges.

They should pinpoint any unsuitable entries and request their deletion. Identity synchronization is used to identify changes in attributes, like phone numbers or department codes, in one system and automatically update them in others. Regardless of the source, all changes must undergo an authorization workflow. This process necessitates validation and approval from business participants before any implementation..

Identity and Access Management as a Process

Figure 2 demonstrates that IAM is a process that creates access roles and corresponding rules based on business processes and security requirements. This system is used to authenticate, authorize, and enforce access to essential resources by creating unique identities assigned to specific roles and governed by rule sets [5].

A process-centric approach may be used by organizations making critical decisions on how IAM should be addressed in their organizations to explain the fundamental ideas. When arranging system criteria and matching functional capabilities to client needs, use it as a template for requests for proposals or bids. Serve as a critical analysis point for discussions with systems engineers and service providers on deployment planning and service rules. Contribute to organizational conversations about sustaining IAM once it goes into production. Identify-current supports process automation tasks. We selected the Report sub-process and control authorization to expand this dissertation [6].

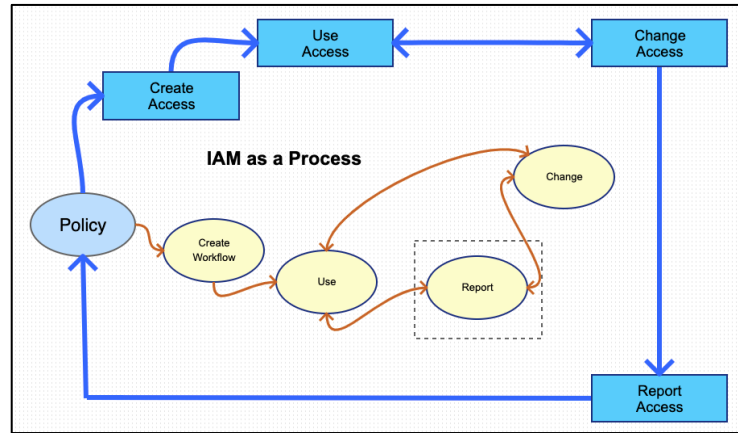


Figure 2. IAM as a Process.

Report Access Model Sub-process of IAM

To ensure compliance and for auditing purposes, having the appropriate tools and capability to create a report on the structure of the model is crucial. Access model reports commonly include identities such as users, accounts, attributes, and orphan/dormant accounts, as well as entitlements such as roles, groups, and accounts. Additionally, reports may cover the history of user activity, role analytics, and workflow data, such as activity within the queue, historical trends, and request popularity. Configuration data, system

data, and troubleshooting information may also be included in the report, such as event logs, unsatisfied requests, and entitlements with no/invalid owners.

The report access model acts as a visual representation of the access "rules," which includes role descriptions, comprehensive rule sets, and contexts. This applies to the resources that identities will engage with, as opposed to the distinct access levels of individual identities. As depicted in Figure 3, the report access model is a periodic process predominantly employed by information security or auditing departments, even though the administration and operations departments utilize this process [7].

To ensure compliance and address any issues, managers and System Access Authorizers should be able to utilize reporting tools. These tools allow them to access user data and generate audits in the event of certain triggers. Processes that may be triggered include violations of Segregation of Duties (SoD) rules, whether for specific users or rules in general. Moreover, violations related to Role-Based Access Control (RBAC) may arise, such as users having entitlements that don't align with their assigned roles due to excess or deficiency. It's equally crucial to keep an eye on access that hasn't been recently or ever certified, as well as denied workflow requests. Additionally, users with entitlements beyond their assigned roles, those with sensitive entitlements, and those whose entitlements result in a high-risk score should be under close scrutiny. Addressing orphaned and inactive accounts is also of significant importance [5].

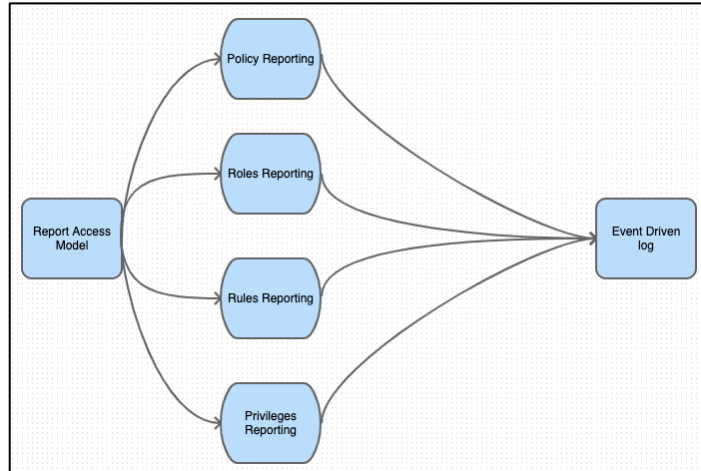


Figure 3. Report Access Model Subprocess.

Report Workflow Sub-process

An IAM must have a very detailed reporting and logging method at every step in the workflow subprocess, as depicted in Figure 4. The reporting and logging will ensure timely identification of a pattern and alerting at the time of an incident occurrence. One example of this method is Approve/Deny subprocess in the workflow. A proper workflow process would promote the unavailability of a resource and create a bottleneck. The Change in the workflow will eliminate these concerns.

The reporting workflow will be utilized to gauge the time taken by the 'Use' workflow subprocess, ensuring that the Service Level Agreement (SLA) is adhered to and making necessary adjustments to the workflow if required.

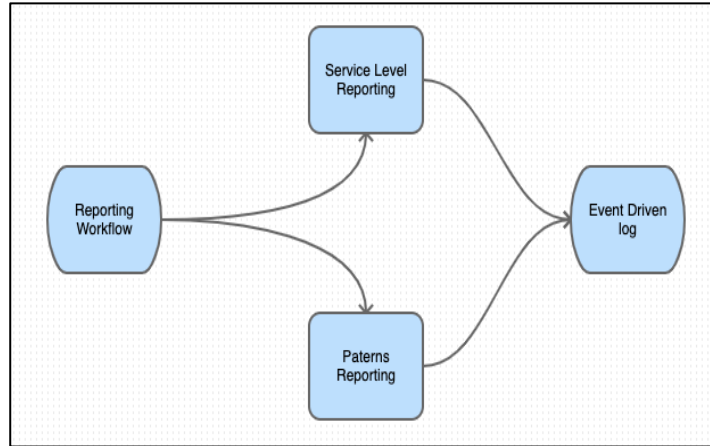


Figure 4. Subprocess of Report Workflow.

In an IAM process, there must be an event-driven logging technique in which a report is generated. To complete the auditing and compliance tasks, the identity reporting will ensure that the reconciliation subprocess is executed within the IAM process at any given time, as depicted in Figure 5. Even the audit and compliance-triggered report will be logged as an event. Some of the reportable audit events are what an individual user has been doing; Users and their access entitlement have been violated; A high-risk termination has been issued; Discrepancies exist in the workflow, known as a toxic combination.

To have a seamless and efficient access and entitlements workflow, enterprises must agree on a single identity source and authoritative body of the IAM process. The single identity source will ensure the changes in identity and entitlement to be centrally governed and act upon should a change occur [8].

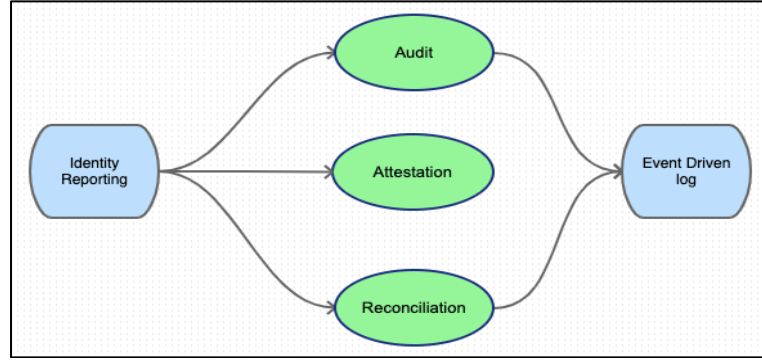


Figure 5. Report Identity Subprocess.

Identity and Access Management Process using Azure Digital Twins

This dissertation initiates a comprehensive introduction to the structure of IoT and its influence on enterprise IAM. It delves into enterprises' difficulties in handling identity and access management in a progressively interconnected environment. Following this, the thesis presents Azure Digital Twins and their significance in IoT, elucidating how it can generate a digital mimicry of IAM processes and lifecycle [8].

Design of IoT Devices and models for Identity and Access Management

The IAM process using IoT devices and models represents a significant evolution in how enterprises manage security. These devices generate large data sets and have unique identities that must be managed effectively to ensure secure access and prevent unauthorized data manipulation. By integrating IoT models into the IAM process, enterprises can create a dynamic and responsive system that continuously uses data feedback from IoT devices to monitor and update asset entitlements. This approach not only enhances security but also allows for the efficient management of the rapidly changing business environment typical of IoT ecosystems. Digital twin technology can further enhance this process by providing a virtual representation of the IAM process, enabling simulation,

analysis, and continuous improvement. Integrating IoT devices and models into the IAM process represents a significant step forward in enterprise security management [9].

Design and Implementation of IAM Process

Treating IAM as a business process rather than a technological challenge has many benefits. The system can be integrated into a workflow, the architecture of the Internet of Things and zero-time operation can be utilized in the process [6], the outcome can be dynamically altered using a Drag-and-Drop system, and the removal of human factors and interaction can help to minimize errors [10]. IAM process, creates, updates, and deletes three kinds of objects that are Identities - records of people and non-human entities; entitlements - grant identities access rights; and Credentials - used by identities to sign into systems – such as passwords, tokens, or certificates. The two latter are the most crucial part which will pin to the identities during inception.

Process control that must exist in any IAM can be categorized as seen in Table 1 below. The objective is to reduce the number of access rights given to a user inappropriately or was never reviewed after it was appropriately given and is no longer needed. Table 1 lists the control objectives that need to be imposed on the IAM process, which was generated based on personal experience and resources. Control objectives are the backbone of an IAM process that must be satisfied [11].

Table 1

List of control objectives in the IAM process

Control Objective	Description
Identity Lifecycle Management	Implement a robust process for managing the lifecycle of identities, including creation, modification, and deletion of user accounts.
Access Rights Review	Regularly review and update access rights to ensure they remain appropriate and remove any unnecessary access.
Strong Authentication	Enforce strong authentication mechanisms to verify the identity of users and systems.
Multi-factor Authentication	Implement multi-factor authentication where necessary to provide an additional layer of security.
Timely Deactivation	Establish a process for timely deactivation of access rights when users leave the organization or change roles.
Audit Trail	Maintain an audit trail of all access rights changes to provide visibility and accountability.
Unauthorized Access Prevention	Implement controls to prevent unauthorized access, such as firewalls, intrusion detection systems, and encryption.
IAM Controls Review	Regularly test and review the effectiveness of IAM controls to ensure they are working as intended.
Compliance	Ensure compliance with relevant laws, regulations, and standards related to identity and access management.

Clearly, a generalized and parameterized framework is needed to encompass and address the complexities of enterprise-wide IAM processes. The access modeling process converts policy inputs, into formal structures like roles and rules. These are subsequently used in the formation and management of identity. This process flow is usually the first to be executed, and it involves the creation and provision of an authentication and authorization model, while also maintaining a formal process trace for future examination [9].

The workflow process delineates how an identity is formed based on existing business policies, like operational procedures. The identity process gets its output from the access modeling process via the workflow. The identity modeling process uses the

workflow to map the required roles, rules, etc., for a specific user. The identity modeling process collaborates with the workflow process to receive suitable mappings. The access modeling process comprises four main processes: create, use, alter, and report. Each process contains subprocesses that offer more in-depth information about the steps required to establish an efficient enterprise access strategy. Although the access modeling process can be carried out in various technology contexts, like a specific corporate application environment, an access management infrastructure program, or an operating system, the process steps are alike. The access modeling approach and strategic information flows are depicted in Figure 6 [12].

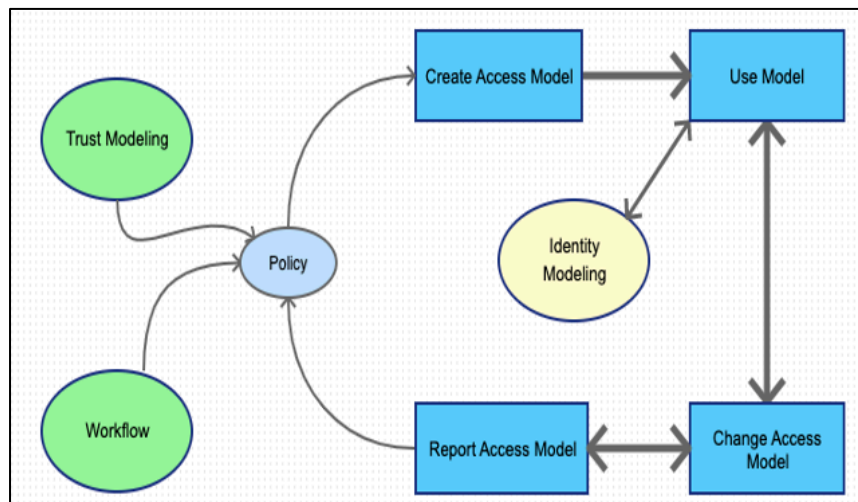


Figure 6. Access Modeling Process.

Digital Twin in Design and Implementation of IAM process.

Digital Twin technology helps in the design and implementation of the Identity and Access Management (IAM) process. By creating a virtual representation of the IAM

process, Digital Twin allows for a comprehensive simulation and analysis of the system. This enables the identification of potential vulnerabilities and inefficiencies, facilitating the development of more secure and efficient IAM processes. In the implementation phase, Digital Twin is a dynamic model that can be continuously updated to reflect changes in the real-world IAM process. This allows for real-time monitoring and adjustment of the IAM process, enhancing responsiveness to the rapidly changing business environment. Furthermore, using Digital Twin technology in the IAM process can facilitate better decision-making, as it provides a detailed understanding of the system's performance under various scenarios. This integration of Digital Twin technology significantly enhances the design and implementation of the IAM process, leading to improved security and efficiency [13].

Azure Technology in Design and Implementation of IAM Process

Digital Twin technology is an innovative method that has been applied in numerous sectors, including the design and execution of IAM processes. This technology entails creating a virtual duplicate of a physical system, enabling real-time tracking, analysis, and optimization. When utilized in IAM, Digital Twin technology can notably improve the security and efficiency of these processes.

In the context of IAM, the Digital Twin can represent the entire network of an organization, including all its devices, users, and access points. This digital representation can provide a comprehensive system view, allowing for real-time monitoring and detection of anomalies or security breaches. This can significantly enhance the system's security, as any unauthorized access can be immediately detected and addressed [13].

Additionally, Digital Twin technology is used to optimize the IAM process. Organizations can detect any bottlenecks in the IAM process and take necessary steps to address them. This can lead to improved efficiency and productivity and reduced costs. The implementation of Digital Twin technology in IAM involves several steps. First, an Azure Digital Twins instance is created, which serves as the foundation for the Digital Twin. Next, resources represent the various components of the IAM system, such as devices, users, and access points. These resources are then modeled using Azure Digital Twins Models, which define the characteristics and behaviors of the resources [14].

Once the models are created, data is imported from a JavaScript Object Notation (JSON) file. This data is used to establish relationships between the IoT Digital Twin Models, which represent the interactions between the various components of the IAM system. An IoT Hub is then created, which serves as the communication gateway between the Digital Twin and the physical IAM system. Devices are added to the Azure Internet of Things Hub, which allows them to communicate with the Digital Twin. An Azure Function App is also created, which is used to ingest the telemetry data from the devices. An event subscription for the Function App is created, and a simulator is used to send telemetry data to the IoT Hub. This data is used to update the state of the Digital Twin in real-time, allowing for continuous monitoring and optimization of the IAM process [13].

In conclusion, Digital Twin technology offers a powerful tool for enhancing the design and implementation of IAM processes. By providing real-time monitoring and analysis, it can significantly improve the security and efficiency of these processes, lead-

ing to improved productivity and reduced costs. As seen in Table 2, we summarized challenges to Identity and Access Management in terms of access and credential, business, and financial aspects.

Table 2

Challenges to Identity and Access Management

<p>Access and Credentials</p> <ul style="list-style-type: none"> • How to request access? • Who must approve the request? • When will the request be completed? • Too many login prompts and passwords. • Orphan and dormant accounts. • Too many people with privileged access. • Static admin and service passwords are a security risk.
<p>Business</p> <ul style="list-style-type: none"> • Regulatory and Compliance <ul style="list-style-type: none"> • HIPAA, Sarbanes Oxley, EU Privacy Directive, etc. • IT Support cost <ul style="list-style-type: none"> • Help desk call volume • Time/Effort to manage access rights • Service Level Agreement (SLA) <ul style="list-style-type: none"> • Faster onboarding • A simple request approval process
<p>Financial</p> <ul style="list-style-type: none"> • An investment in identity and access management processes and infrastructure is typically supported by <ul style="list-style-type: none"> • Cost savings <ul style="list-style-type: none"> • Reassign staff out of the help desk or user administration group. • Improved productivity <ul style="list-style-type: none"> • Help new users start work sooner and eliminate delays experienced by users who have problems or need changes. • Stronger security <ul style="list-style-type: none"> • Clean up entitlements, enforce security policies, and create audit logs. Comply with SOX, HIPAA, etc.

Design and Implementation of IAM Processes

The design and implementation of IAM processes require an engineering approach. The proper engineering approach in these cases is called the engineering of enterprise systems. The discipline that applies systems engineering to the essential part of businesses is generally called Enterprise System Engineering (ESE). Enterprise engineering differs from systems engineering because it considers an organization's whole lifecycle. Businesses have existed for periods, but they have not always been designed similarly. Furthermore, firms were not previously considered as a whole system that could be logically designed, depicted in Figure 7 [14].

Enterprises have expanded rapidly with little attention to the systematic design of their processes and systems. It has been indicated that many organizations' business procedures were created long before information technology existed. It is not surprising that some companies and organizations still have business processes that model an IT system mechanism. Many small modifications were made over time without considering and looking at the enterprise processes as a whole [15].

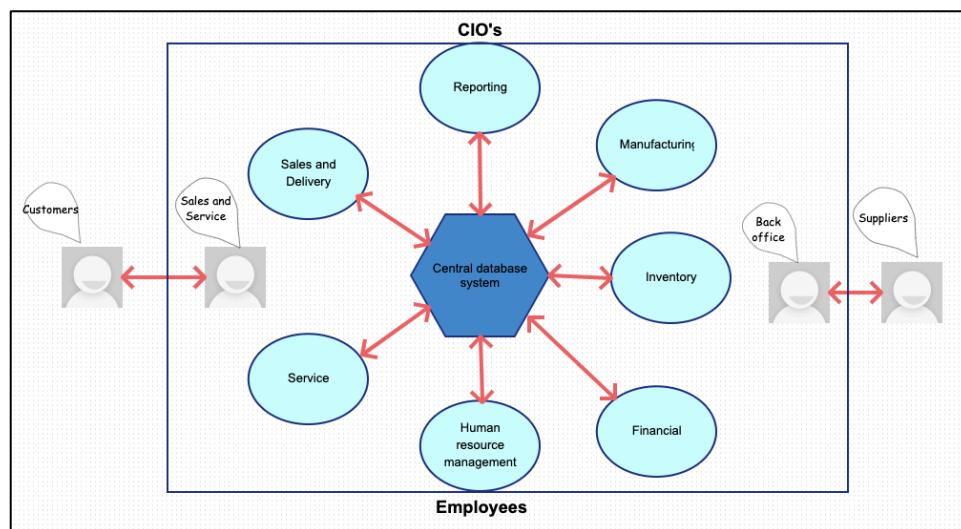


Figure 7. Enterprise system.

Enterprise Systems and Enterprise Architecture

A business is a social system that links people, information, and technology resources to achieve a common goal. Many interactions are required to understand enterprise behavior as a system. Sharing information, coordinating functions, and allocating resources are just examples.

The term 'enterprise' encompasses a range of entities, including governments, supply chains, businesses, and virtual enterprises. Moreover, the term 'organization' is restricted as it represents one aspect of the enterprise. Enterprise Architecture is a discipline that, i) outlines, organizes, standardizes, and records the complete architecture and all significant components of the respective organization, spanning relevant domains such as business, digital, physical, or organizational; and ii) the relationships and interactions between elements belonging to those domains, such as processes, functions, applications, events, data, or technologies [16].

To ensure efficiency in the lifecycle of workflow problems, the enterprise needs to provide an overall enterprise design to control all its projects. Even a small enterprise project should conform to the enterprise architecture. In this way, the enterprise will ensure that all projects get the attention they need, which should accomplish its goals. Many organizations are currently using enterprise systems, a collection of complicated components that collect and deliver data to aid in the decision-making required for corporate control. Successful organizations use the enterprise system to make accurate decisions in a short amount of time, allowing the management to make the best decisions for the company [17].

Process Improvement Using Quality Definitions

Process improvement refers to identifying, evaluating, and enhancing existing business processes within an organization to meet new standards or quality requirements. Quality is one of the most crucial elements in engineering. Basic defaults aid us in attaining exceptional quality. Concentrating on quality attributes recognizes that quality is multi-faceted and consists of conflicting elements. Quality attributes encompass aspects such as durability, usability, and user-friendliness, among others [17].

Some fundamental qualities of quality must be included in the product, such as look, cost, and dependability. The consumer must be satisfied with the degree of quality. Otherwise, the product will be overlooked. In addition, when quality is at the necessary level, it may rise, making it less competitive in the marketplace. It is difficult to locate a product that meets all the quality criteria. In addition, the importance given to each quality criterion defines the aspect of quality. When contrasted to a quality profile, emphasis is useful in deciding between product alternatives. Table 3 shows a range of quality emphasis for watch quality. Generally, quality is the capability of a product concerning a specified number of quality attribute emphases [18].

Table 3

Spectrum of product quality attribute emphasis

Emphasis/ Quality Attribute	Appearance	Cost	Reliability
None	X	X	X
Reliability Emphasis	X	X	√
Cost Emphasis	X	√	X
Cost / Reliability Emphasis	X	√	√
Appearance Emphasis	√	X	X
Appearance / Reliability Emphasis	√	X	√
Appearance / Cost Emphasis	√	√	X
Excellence	√	√	√

Achieving Quality Improved Process

Isolated and static views are two ways to evaluate process capabilities and product quality. The definition of process focus is attaining quality through a process analysis that begins with adding descriptions of the process's actions to a logical and understandable form. The process should also be specified in the process model since it may be used to enhance the process. A model, in general, is useful for expressing the most important elements of reality while allowing for limitless detail [18].

All subsequent process improvement efforts may be thought of as being built on the basis of process modeling. The capacity to comprehend and enable objective evaluations, software processes, and making changes are three characteristics that make process modeling valuable. Process modeling can also aid with model evaluation by providing

recommendations for process changes. When comparing processes, the method of calculating quality leverage metrics may be used. The work of process improvement has a vision. This process follows the vision of improvement because improvement is continuous. The iterative technique of defining the process, execution, measurement gathering, improvement, and analysis are all emphasized in this view of process improvement. As demonstrated in Figure 8, each stage has the essential input for the next phase in the process.

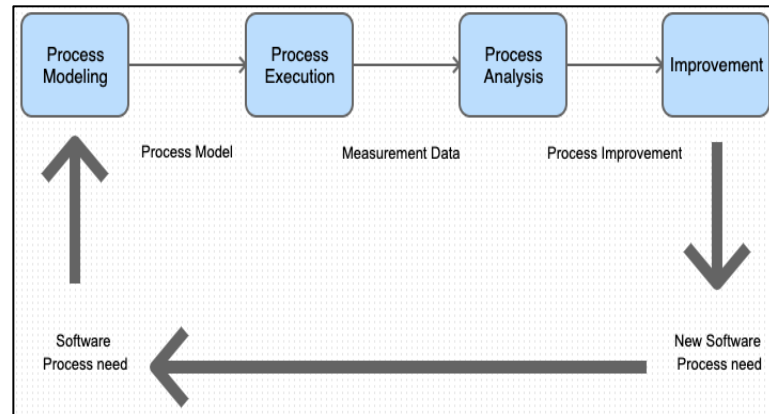


Figure 8. Process-centered improvement operational concept.

A Context for Process Improvement

The connection between the evolutionary activities in software processes and the activities that support those processes determines the components we need to describe process-centered improvement. The process improvement group oversees the organization's software development. The process improvement group is also a valuable resource for many businesses. The process technology group strives to enhance the tools and techniques that support the process. Figure 9 depicts this connection.

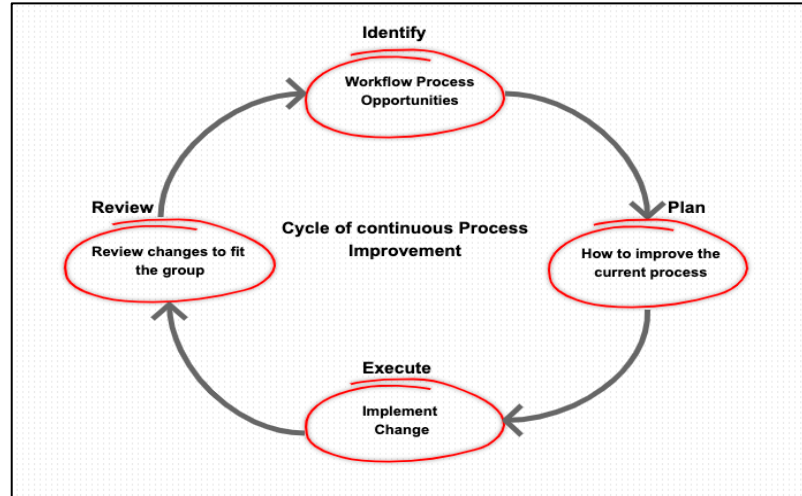


Figure 9. Process-Centered engineering quality model.

Collecting data and evaluating obtained data are three integral components of process improvement. Although the automated tools make the process easier, some sub-functions will still require human participation. Additionally, process modeling can be built using the tools provided by the data provided by the engineering activity. The feedback gained through actual process modeling execution may be used to help the organization design process technology and improve it. The function of process improvement can be improved through process recording and model simulation done using Azure Digital Twins [19].

Process Improvement Activities and Modeling

The concept of process improvement activities will be explained in this section. Lifecycle process engineering has also been defined previously. For the process improvement connection to be established, all activities here must work on specified objectives within the context of process improvement and the extra requirements of infrastructure and communication demands. Furthermore, communication and infrastructure capabilities can aid in developing each level of activity dimension.

Technology acquisition, process modeling, measurement and execution, and analysis and improvement are the four components of the process-centered improvement operations approach, as outlined. The latest advancements in process modeling languages that have emerged from the opinions of the underlying methodologies will be discussed in this part. By creating a set of criteria for each formalism, we can tackle improvement challenges differently. The formulation of process support tools is completed by a significant amount of research, which includes the construction of process models as the foundation for automating development environments [20].

In order to make the process transparent, process modeling is required. Process modeling, for example, provides the organization with a number of essential issues such as the resources and capabilities required to begin the activity, the scope of the process to be modeled, the features of reality that must be included, and the components that should be removed. There are also some benefits to using the process to think about and determine the contents of the process model. Traditionally, the concept of the IAM has always been synonymous with technical tasks and challenges. Such a task would have included but not limited to either purchasing a software system from a vendor and have them deploy enterprise-wide or a few technical staff install it. Although this method is widely being used to this day and has been mostly the choice of the companies, but some have been trying to develop their own IAM software which is wished to perform as well as a very costly third-party software package [20].

In either of the above cases, there are many common problem and mishap which is often overlooked, and not enough attention is paid to. Some of the burdens which can be problematic are, lack of automation, lack of dynamic changes, the operator with access

is on vacation, silo system of IAM which know nothing about each other, and many more.

IAM systems creates, updates, and deletes three kinds of attributes, Identities - records of people and nonhuman personas. For example, how the records are entered in database tables. Entitlements - which grant identities access rights. For example, how users are authenticated before they are granted access to sensitive data or functions? Credentials - used by identities to sign into systems -- such as passwords, tokens, or certificates. For Example, is there a close link between what users can access on systems and applications and their real-world business responsibilities?

In other words, identities are managed just so that there will be something on which to pin credentials and entitlements. The objective is to maximize the security expressed in authentication and authorization decisions while minimizing the time and cost spent managing the underlying identities, entitlements, and credentials. It's not enough to examine existing identities to find users who have entitlements that are not appropriate to their business needs. Therefore, new entitlements should be assigned to be consistent with business needs; entitlements should be revoked in response to business changes, such as terminations or transfers; and when analytics find inappropriate entitlements, they should be deactivated as quickly as possible [19].

Components of Enterprise Design and Lifecycle

The designer is the most significant component of the "business," especially if it is a product [12]. During all stages of the design project, the designer cannot design an

enterprise alone; instead, numerous people from various disciplines are frequently involved. Some of the members of the enterprise design team have job titles assigned to them as the following [14, 15].

Business Systems Analyst has extensive knowledge of the business domain language, such as accounting, and the technology employed in that domain. Enterprise Architect is a professional with extensive experience building a comprehensive perspective of an organization's strategy, information, procedures, and structure. A system Architect is A person with experience in designing a high-level design of technical system and is referred to as a system architect. A project Manager is a person who oversees the fulfillment of all project objectives. The project manager is in charge of identifying all team members, supervising them, planning the project, tracking its progress, and being accountable for all project outcomes. A system designer is a person who has experience in designing one or more parts of the system. This person has the technical background to know how the system will work. The change Manager is the person in charge of the change management strategy, this person is in charge of ensuring that the change management plan is carried out successfully. A system Engineer is a technical expert with knowledge of the entire process of creating, running, developing, maintaining, and replacing quality management systems. An application Developer is a programmer with experience developing new software depending on the technology [21].

The enterprise lifecycle is divided into three stages, development, deployment, and operation. Development is essential since it encompasses the engineering steps required to build an enterprise system. The deployment process encompasses the enterprise

system's change management procedure. The administration of the overall enterprise system is referred to as operation. The typical phases of an enterprise's lifecycle can be summarized as System identification, Analysis, Design, Construction, Implementation, Operation and Maintenance, and finally, Decommission. Figure 10 shows that an enterprise may cycle through many development projects before it is decommissioned.

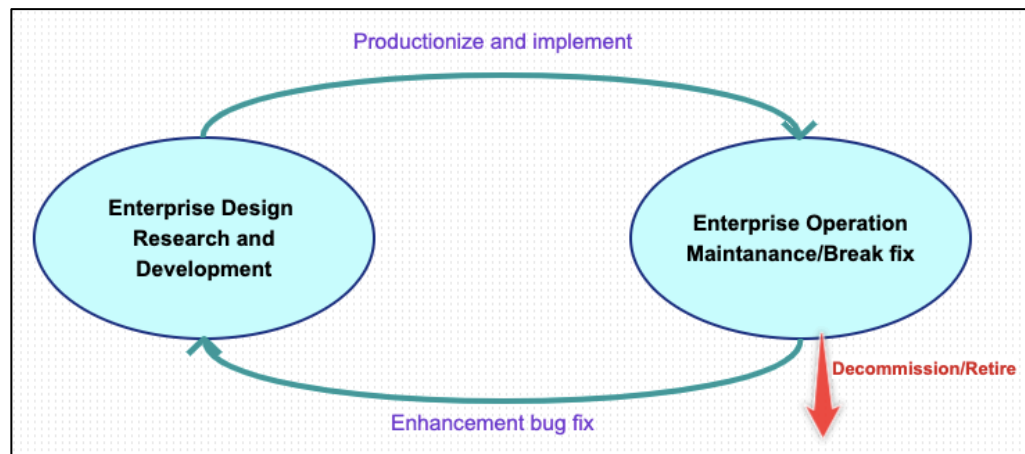


Figure 10. Relationship between enterprise development and operation.

Engineering of Enterprises

There are a variety of viewpoints on the importance of enterprise engineering. Davenport and Short, for example, discuss the need for engineers who can design and analyze business processes and apply IT to improve them. It also explicitly asserts the necessity for an engineering approach and describes enterprise engineering as a set of change management techniques [21].

It is emphasized the necessity to transition from a haphazard approach to enterprise design towards a methodical, engineering-based approach. Three foundations are

required for an enterprise engineering discipline. Enterprise integration knowledge outlines the various ways in which the different elements of an enterprise can be organized and integrated to function collectively.

Enterprise engineering concentrates on the design of the entire enterprise. Businesses can be categorized into two types for engineering purposes. With the rise of the service industry, agile businesses should adopt a service-based process model. Overall, it's becoming evident that a rigorous engineering approach is needed for designing integrated enterprise solutions. Enterprise architecture serves as a strategic foundation for enterprise information systems. This principle outlines the facility's vision, the information and methods required to realize it, and the evolutionary processes for incorporating new technologies in response to changing vision requirements, as illustrated in Figure 11 [11].

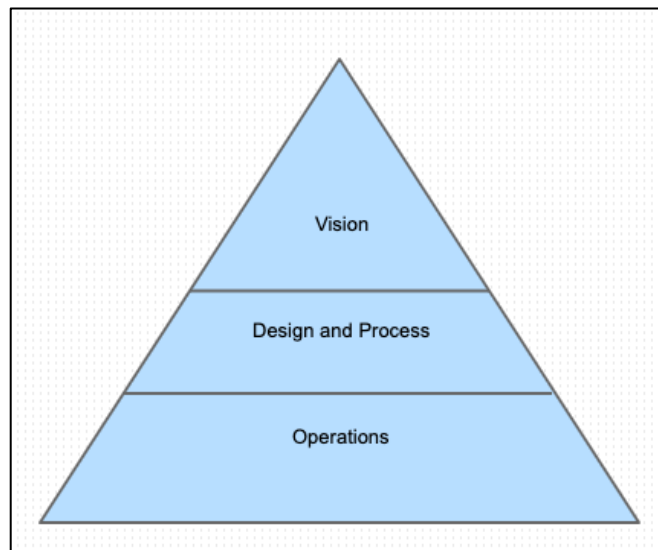


Figure 11. The three layers of enterprise architecture.

Layers of enterprise architecture include Vision includes the goals, objectives, and strategies of an organization. Design and Process include the Research and Development

global roll-out plan; and Operations include day-to-day production run, break-fix, and updates. Advancement in Azure Digital Twins allows efficient simulation of Enterprise Architecture.

A Process-Centered Engineering Quality Model

The quality profile that has been created concludes that engineering design quality factors such as timeliness, cost, and reliability should drive the design process of engineering artifacts as defined by [11]. Engineering design quality criteria must assume that the customer has a budget for the product; other attributes include the time required to complete the product and the product's minimum operational requirements. Table 4 shows a quality profile based on engineering design quality factors. The expense of developing the product has exceeded estimates. As a result, the supplier's cost will be lower than the customer's overhead cost barrier. While timeliness is commendable, the process that supports product creation has fallen short of reliability requirements. Table 4 shows a spectrum of concentration that can be applied to engineering design [11].

Table 4

Spectrum of engineering design emphasis

Emphasis/ Quality Attribute	Cost	Timeliness	Reliability
Pure Research	X	X	X
Reliability Emphasis	X	X	√
Timeliness Emphasis	X	√	X
Timeliness / Reliability Emphasis	X	√	√
Cost Emphasis	√	X	X
Cost / Reliability Emphasis	√	X	√
Cost / Timeliness Emphasis	√	√	X
Pure Engineering	√	√	√

Instead of focusing solely on product quality, the concept of quality can be broadened by comprehending the processes that underpin product construction. These process features enable for product and resource transformation. This process involves a number of interconnected tools, people, and activities. The process-centered engineer quality model encompasses all interactions within the engineering activity as well as their natural processes. The process is simulated in Azure Digital Twins.

A General Systems Design Methodology

In this section, we present the first composite view of a design methodology that attempts to combine the desirable attributes of several of the design strategies. The basic structure of this methodology has its origins in a design approach referred to as edges-in design, which is based on design principles. The basic edges in the design approach,

along with its supporting principles, has been extended somewhat to reflect the requirements identified. We have also attempted to form a methodology amenable to partial automation. Included are specific attempts to acknowledge the impact of preconceived concepts of the final implementation hardware, the requirement for accommodating constraints, and the need for iteration within the logical design process to accommodate the possibility of multiple virtual machine layers in complex system designs.

We present an overview of the entire design process aimed at establishing a framework within which the detailed discussions of the remainder of the book are set. We shall not go into the details of specific tactics and mechanisms in this section. Still, we shall relate a design philosophy to a design strategy such that the rationale for each phase and step of the methodology is clear and the relationship of the strategy to the general system perspective is firmly understood [12, 13, 14, 15].

The methodology structures the approach to this problem and provides a strategy that can be followed to arrive at good designs. It contains separate phases that partition the logical design from the physical. The logical design phase may be iterated through several logical layers until a suitable level for mapping to a specific hardware/software boundary is reached. The starting point for each iteration through the logical design phase is a set of virtual machine requirements. This partitioning of the logical system description into a hierarchy of logical structures facilitates the design of complex systems by introducing a simple mechanism for modularizing the design. Furthermore, as we shall see in later chapters, the concepts of nesting packages and tasks in a structure diagram description of the system directly support such a hierarchical design approach.

The basic philosophy of this design approach is that we should first determine exactly what the system must do before considering how it is to be implemented. This implies both a careful analysis of user requirements, constraints, and specifications and a complete definition of the logical system structure prior to actual hardware considerations. By imposing a hierarchical structure on the logical system description, the problem of partitioning and allocating this description to a special hardware/software combination can also be approached in a structured, modular fashion [16, 17, 18].

Edges-in Design

We begin this section with an overview of the basic edges-in-design approach. We then consider extensions to this approach to form a design methodology based on a multileveled system perspective and show how this perspective is reflected in the structure of the methodology. To effectively design a system based on the requirements provided, it is recommended that the designer initially distinguish between nonfunctional requirements and constraints. Following this, the necessary system functions should be identified, starting with the system interfaces or boundaries. To achieve this, one must experiment with divisions. From these interface descriptions, the suggested procedure follows a top-down partitioning and refinement of the system's interior to create a logical description of all system functions necessary to support the required system operation. This implies using some descriptive mechanism to develop a logical software architecture description. In the original development of the methodology, the descriptive mechanism used was access graphs. From the access graphs, the designer attempts to define all the necessary active processes for the system and then define their interaction. The descrip-

tion of process interaction is accomplished by defining a set of monitors to handle scheduling and synchronization. The supporting semaphore structures are assumed as kernel functions. Alternately, the structure diagram mechanism could be used, based on defining tasks that interact through rendezvous mechanisms [19, 20, 21, 22].

Once the logical software architecture description has been completed, the next step is to identify the necessary support functions required for implementation. If an access graph description is used, then support mechanisms must include a set of kernel functions that provide semaphores, process context switching, and support for the appropriate process/monitor communications, including call routing and parameter passing. Thus, the kernel features, in addition to the necessary support of a programming language to implement the software, represent the virtual machine that is needed to support the implementation of the logical system description. Having defined this logical architecture, including the virtual machine support requirements, the designer is ready to face the problem of partitioning this logical system description and allocating the various partitions to a specific hardware architecture for implementation.

The kernel functions were defined as a special critical region of software or, possibly, as hardware functions, such as a hardware test and set operation. The kernel functions and the hardware, therefore, offered an instruction set suitable for the implementation of the logical architecture of the system. The partitioning and allocation problem centered on the partitioning of the logical architecture of the software and allocating it to the various processors while maintaining the kernel functions as a common critical region. It is important to note that, in this allocation process, the basic concept of hardware

architecture is either a conventional uniprocessor or a collection of interconnected processors. This assumption was entirely justified in the original development of the methodology since the fundamental processing elements were always microprocessors.

The benefits of following this approach to the design of multiple microprocessor systems are substantial. By defining the complete logical structure of the software in advance of the hardware, the problems of hardware/software integration are considerably reduced. Because the complete software structure is known, the selection of the hardware can be done based on reasonably well-defined performance requirements. Indeed, the complete software structure can be implemented on a general-purpose computer by well-known multitasking methods to verify the logical operation of the design. Comparative performance measures can be made for various proposed system partitions prior to the actual construction of the final system hardware [23, 24, 25].

Multileveled Systems Design

We now present an extension to the basic edges-in-design approach. This extension is based on an abstraction of the conventional onion-skin system model to account for multiple logical levels for which the final implementation, as either hardware or software, is not preconceived. These levels represent successive logical layers of a hierarchical system organization. Within a system, distinct modes of operation can be seen as divisions in a logical layer, which create multiple virtual machines needed for the same system [25].

This logical view of systems represents, to some extent, a preconception regarding the final system structure. Indeed, by focusing on this system perspective, we are explicitly acknowledging this as our fundamental starting point in the system design pro-

cess. In a specific system design problem, this general model of system structure is refined and modified by the specific requirements and constraints imposed on the design. Given the requirements specification, the first phase of the design process is to fit the stated system requirements into a logical system description based on this system perspective.

The unique aspects of this expanded design methodology that differentiate it from the basic edges-in approach include the explicit recognition of the influence of preconceptions about the final system's overall structure, the likely sensors technology for implementation, and the rapid evolution of integrated circuit technology. It also involves the introduction of an initial system partitioning activity as part of the first step of partitioning system functions and constraints [24]. There's an iteration of the logical architecture and virtual machine definition steps in the logical design phase that breaks down the overall logical system design into a hierarchically structured, modularized set of smaller logical design problems. It also acknowledges that the logical partitioning and hierarchical structure of the logical system design might not align with the physical partitioning and topology of the final system, necessitating a structured method for partitioning the logical architecture and allocating various portions of the system to hardware at different levels of the logical design. The abstraction we are proposing to the basic onion-skin system model extends the virtual machine concept to all system layers regardless of whether they will eventually be implemented entirely in hardware or software or as some combination of both [16].

The definition of a virtual machine is an outline of the system's visible attributes, which are relevant at that particular level of the system's description. Thus, we consider

each layer of the system to be representable as a set of logical primitive operations that summarize the observable functions of the system at that level, not distinguishing between those functions executed directly in hardware and those executed in software. This lack of commitment to either hardware or software at all logical levels of the system description is necessary if we are to create a consistent logical description that will later be partitioned to define a specific hardware/software boundary for the final implementation. Indeed, the blurring of the hardware/software boundary mentioned has reached a stage where the distinction between the two is often determined as much by the observer's point of view as by any fact of implementation [24].

The design approach we are presenting is based on the concept that it is unnecessary to consider system functions as either hardware or software until a complete logical description of the system has been created. Hence, the initial driving force behind the design process is not the designer's knowledge of hardware structures. However, this becomes very important in later stages of the design process, but an understanding of the system requirements and the user's perspective of the system and its operation. The user-visible system interfaces and observable modes of system operation guide the initial system division into virtual machine levels and the identification of the necessary functions at each level. From this perspective, the edges-in approach can be seen as a variant of the outside-in design strategy [26].

The second stage of the design process essentially mirrors the fundamental structure of the edges-in approach. The expansion here involves the specific incorporation of an initial division of the system into a that corresponds to the virtual machine layers of

our general system model as part of the initial system function identification step. The details of the logical architecture are defined iteratively through successive refinements, starting with the highest-level virtual machine interfaces and then for each sublevel until all user-visible system features have been outlined. This method divides the overall logical structure of complex systems into several smaller, more manageable logical design problems. [25, 26].

The third phase, selecting a suitable hardware architecture, may also be approached hierarchically in many design problems by carrying out the partitioning and allocation of major system elements at a high logical level. This is often necessary due to specific constraints or physical partitioning requirements stated in the specification. Once the specific functional characteristics of these major system partitions have been defined, including their performance requirements and interfacing details, they can be treated as separate subsystem design problems.

The iterative process of logical design should be continued through at least all user-visible system levels before committing to major hardware architecture features. This will help to prevent costly and time-consuming redesigns and retrofits that can occur when a hardware commitment is made before all the details of the logical system structure have been worked out. The final phase of the design process, component selection and final implementation of the design, comes after the logical structure of the system has been completely determined, various partitions and allocations to hardware have been analyzed, and a final hardware architecture has been selected. Here, the final decisions on the hardware/software boundary will be made as the detailed structure of the various hardware elements is fixed [26].

Requirements Analysis and System Specification

As with all the other design approaches we have discussed, the approach advocated in this section begins with explicitly recognizing the need for a thorough problem analysis leading to identifying user requirements and constraints from which a system requirements specification must be generated. The issues of requirements analysis and system specification are considered in detail in this section. First, however, we stress that it is in this initial phase of the overall design of systems that many design efforts have yet to go astray. Regardless of the ingenuity and expertise of the design team, if a clear understanding of the system requirements and constraints has not been gained, it is doubtful that the final design will meet them [27].

Hence, a key point of the entire design philosophy is that it begins with, and is based on, a clear understanding and a concise specification of what must be designed and what constraints are placed on the design. This in turn tends to focus the designer's initial perceptions of the system structure in terms of what is perceived to be possible within the constraints. These initial perceptions of system structure will depend, to a large degree, on perceived component technologies with which the final system will be implemented. When creating specifications for processing systems in an environment of rapidly changing fundamental component capabilities and cost/performance ratios, it is possible, even probable, that one of two general problems will result. First, if the user's perception of what is possible does not reflect the current capabilities, then the specifications may be overly restrictive and limit the designer in achieving an optimal design; on the other hand, the perceived capabilities of current technology, on the part of the specifier, may be

overly ambitious and lead to specifications that cannot be met within the performance requirements and constraints imposed. Thus, it is very important to examine the specifications carefully to determine how realistic the overall requirements are before beginning the design process at all [27].

Requirements Analysis

Ideally, the design process begins with a clear specification of system requirements. The specification should provide a concise statement of all necessary functional and nonfunctional system characteristics, including any implementation constraints, and minimum throughput requirements, that must be met. The system specification sets out both the quantitative and the qualitative characteristics on which the acceptance of the final system design will be based. In many situations, as viewed from the perspective of the design team, the formulation of both the user requirements and the resulting system specification is considered a customer responsibility. In such cases, the system specification forms the initial input to the design process and represents the baseline for validation of the various levels of system description that arise throughout the design process.

The essential requirement for a suitable system specification is that it provides a clear statement of all system features required by the user in a form that is accessible both to the user and to the system designer. This document represents the interface between the user and the designer. While it is not reasonable for the designer to expect the system users, in general, to be conversant in the concepts of systems design, it is almost always the case that the users will expect the designer to be highly knowledgeable in the user's application area [28].

There are many kinds of specifications and formats for specifications. The basic purpose of any specification is to state concisely the significant properties or features of a system or object. Precisely what properties or features are considered significant will vary extensively, depending on the kind of specification and its intended purpose. In general, we can divide specifications into two broad categories. Requirements specifications; and Implementation specifications. Requirements specifications are generally intended to outline those properties and features required in a system as well as any implementation constraints that are to be imposed. Still, detailed implication features and non-user-visible system characteristics are left unspecified. Essentially, a requirements specification assumes that some form of design and/or development work is required to reach an implementation that satisfies the requirements. Generally, some identifier should always be used in conjunction with the word specification to indicate the nature of the object being specified and the purpose of the specification, such as Functional requirements specifications, System design specifications, System development specifications, Functional item specifications, and Program development specifications [28].

Specifications generally state the detailed characteristics of the actual implementation or fabrication of an object. The design process may be considered as creating an implementation specification, which guides the detailed fabrication of the final physical product. The design process usually involves initial prototyping to validate the design. It is the detailed specification of exactly how the implementation of a set of requirements can be achieved that is the real result of the design process. A requirements specification states what is required but not specifically how it will be achieved. An implementation

specification details exactly how some object is to be created. Since we are interested in design, we shall consider only requirements specifications in this section [27].

Two preliminary steps are necessary to prepare specifications. First, the problem must be characterized, and from this, a set of user requirements must be prepared, second, from these requirements, specifications are prepared that include the circumferential constraints, which the system must satisfy, in addition to the operational features. The final document can occupy volumes and require technical experts and legal assistance to prepare and interpret. We characterize and quantify some aspects of specifications regarding their logical structure.

The importance of initial requirements analysis is often under-stressed in discussions of design methodologies and approaches. As stated previously, it is often assumed that requirements specification is a customer function, while the actual specification is, in fact, the interface between the designer and the user. It sets out the functional requirements and constraints on which final acceptance of the system must be based. Thus, validation of the design against the requirements specification must be carried out throughout the design process. A central theme of our design methodology is that it begins with and is based on a clear understanding of what must be designed and what constraints are imposed on the design [28].

As shown in Table 5, Table 6, and Table 7, we listed current issues, deficiencies in current approaches, and addressing the needs, respectively. As such, we conclude our literature survey by listing the needs. In the next chapter, we will start with our approach to the design and implementation of IAM processes.

Table 5

Current issues

- There are many IAM solutions available.
- Each solution addresses the IAM issues differently.
- While some provide enhanced features, they increase complexity and operational cost.
- Some solutions focus on automation, whereas others focus on workflow processes.
- Security is the primary concern where ease of use and flexibility is not considered.

Table 6

Deficiencies and current approaches

- **Functionality and features:** While many IAM solutions are available, they primarily provide the same features.
- **Workflow and automation:** Most of the solutions lack seamless workflow automation.
- **Toxic Combination:** They mostly ignore insecure combinations, such as the requester is also an approver of their requests.
- **Enhancement:** There is little or no room for infrastructure enhancement or expansions.

Table 7

Addressing the needs

- Ease of use: How simple is it to set up and manage users, access rights, roles, etc.?
- Integration capabilities: How well can the system integrate with your existing infrastructure?
- Compliance and Reporting: How well does the system assist with compliance and reporting?
- Scalability: Can the system grow with your organization?
- Security: What security features are built-in?
- Cost: What is the pricing structure?

CHAPTER 3

SYSTEM DESIGN AS A STRUCTURED PROCESS

System design is a structured process that requires two essential conditions for success. The designer must possess a guiding technique or algorithm to navigate the design process, and they must have a thorough understanding of the requirements and the correct perspective on the systems to be developed. The design methodology should detail the steps that transform the specifications or needs statement into the final product, considering all constraints and operational requirements. It is difficult, if not impossible, to measure the extensive knowledge and experience that a sturdy design brings. Nevertheless, it is possible to analyze and explain the general framework of their methodology. It's also feasible to chart the sequence and interconnections between certain key processes that consistently yield successful systems [24].

The most vital elements of a successful design methodology often involve principles that assist individuals in making decisions amidst uncertainty. A viewpoint on what is being developed is also essential to the design process. Although it is frequently disregarded, our perspectives limit our ability to succeed. As a result, having the right perspective and using complementary design methodologies are crucial requirements for innovation. We see that techniques frequently appear to be very diverse in detail because different procedures are needed to adjust the methodology to a specific perspective. But it's astonishing how many design techniques share the same basic structure. The exploration of design techniques is the focus of this chapter [29].

A General System Perspective

Before discussing design techniques, it's important to clarify an overall viewpoint of the system that could be the focus of our design work. Understanding that there are always two models, one at the logical level that describes what the system does or must do and one at the physical level that specifies how the logical components are to be distributed or realized in terms of some combination of IoT sensors and software, is also essential.

Virtual machines, as in Digital Twin, serve as logical interfaces, to sum up. Depending on elements like the number of tiers of system programmability or user-visible system interfaces, the number of virtual machine layers in each system may change. Each virtual machine layer gives the subsequent higher layer a logical system view. For various reasons, the hardware designers' physical view does not necessarily need to match this logical view. One of the functions of the virtual machine is frequently to make the hardware invisible to the programmer or the system's final user. We can already observe the integration of functionality to enable several logical system levels with sensors [29].

An entirely partitioned onion represents the system's overall logical structure. In a true sense, this is the systems designer's first significant contribution because it creates the framework for how the rest of the design will be implemented. In the following parts, we'll go through how these levels relate, the problems with partitions, and the design strategies for implementing each. However, before moving on, there are a few terms and terminology related to design in general that will help create a shared understanding. The design team should communicate their opinion on the project's outcomes, an essential

quality. Without a shared viewpoint, the team would almost certainly encounter communication issues, which could result in failure in the worst-case scenario or exorbitant expenses in coordination overhead in the best-case scenario.

The role of a design methodology is to identify the structure and components of a system that meet the given requirements [30]. These requirements are derived from an analysis of the initial problem and are used to formulate the specifications. Ideally, the design process should not be constrained by these specifications, meaning the system's structure should not be pre-determined. However, this is seldom practical. Specifying a system without some commitment to its structure is challenging, if not impossible. As a result, when the design phase begins, there are almost always some implicit structural limitations. Constraints from the implementation phase also impact the design phase, restricting some options. The primary differences in methodologies can be traced back to how these constraints are recognized and interpreted at each stage.

Structural Attributes of Design Methodologies

There is a wealth of material surrounding the general area of design that is difficult to assimilate without some structure. Unfortunately, many, often conflicting, structures attempt to organize the many approaches. We shall begin with a simplistic view with IoT perspective of the design process, and develop our own point of view [31]. Four major activities are isolated into the specification, logical systems design, sensor architecture selection, and system implementation. This is where using Digital Twin and Azure implementation pays off.

The process of specification activity entails transforming user requirements into a document that accurately conveys the essence of the problem to be tackled. This document serves as a guide for the design process and also enables quantifiable assessments of the design outcomes. The design process starts and finishes with the specifications. During the logical systems design phase, the specifications are analyzed and a group of virtual machines are created. These virtual machines can eventually be transformed into a mixture of IoT sensors and software.

Internet of Things Sensors Architecture

In this step, the virtual machines are allocated to candidate IoT sensor architectures. Here, the problem of partitioning the logical design for eventual implementation must be faced. This stage of the design demands knowledge across several disciplines. To create the necessary system, the implementation stage requires careful consideration of the hardware components and the development of both the hardware and software aspects, with attention paid to all the details involved. A nontrivial aspect of this process is the final validation demonstrating performance following the original specifications. When approached consistently, this stage can often be reduced to several smaller-scale iterations of the design stages [29].

There is, in general, very little disagreement with this overall view of the design process if the discussion of details proceeds no further. It is in the details of how the steps are carried out and on the principles that influence decisions that disagreement is found. Some views are fundamentally different, and it is important to understand these differences and why they occur. A design methodology possesses the basic attribute of providing order to and a reason for each step in the design process. Usually, the methodology

and the reasons apply to a broad class of applications and, therefore, must be understandable and consistent when applied across the class members.

When creating major design components, it is important to adhere to a design strategy, which outlines specific steps and a general approach to the process. Two categories of strategies have been suggested to aid design methodologies [31]. Various design strategies exist, including orthogonal and hybrid approaches. Orthogonal strategies consist of top-down and bottom-up methods. In contrast, hybrid strategies focus on critical components first, starting from the outside-in or inside-out, or beginning with edges-in or interfaces first. Although there is some overlap among these strategies, we will discuss them in greater detail later. Design tactics are specific procedures for implementing steps in the overall strategy, and they may be either controlled or uncontrolled approaches. The selection of the appropriate tactical approach at the appropriate stage of the design process is a major issue in the design. Tools called design mechanisms are used to help with certain parts of the design process, whether it's strategic or tactical. These mechanisms can include things like diagrams, graphs, and tables, and they help represent the design process itself. Pseudo-code and other notational conventions may also be used.. As observed earlier, mechanisms often disguise the basic equivalence of two methodologies, and their details are often the most difficult to understand. A design methodology can be examined from a more global point of view [30]. These often contain opposing attributes regarding acceptance by the various groups in a particular environment.

Structural Attributes

Every methodology can be seen as having a two-part fundamental structure. The first part involves a series of steps, accompanied by a set of guiding principles, which are

followed to create a design. These principles assist in decision-making during uncertain situations. To put a plan into action, the next step is to make design choices that match a set procedure.

The characteristics of a methodology encompass a variety of attributes that are separate from its fundamental structure. These features often rely on the specific mechanisms utilized and are typically the most influential factors in selecting a methodology from various options. For instance, the management of a design project imposes its own set of requirements, such as the need for benchmarks and progress reportability, the ability to predict and therefore organize the use of design-related resources, including personnel, and the necessity for documentation [32]. Satisfying these requirements depends on how stringent they are in a particular environment and on the basic features of the design mechanisms used in the methodology. To some extent, it is the mechanisms that yield the most effective interfaces between the structure and the features perceived by both management and the designers; it is the mechanism that represents the medium for communicating the design. A point of some importance, therefore, is the proper choice of mechanisms. The associated mechanisms can obscure a good methodology, and, to some extent, a poor methodology can be made acceptable with suitable mechanisms. Finally, in this regard, the structure and features of a methodology may be so inextricably combined as to be inseparable even for examination. It is desirable that the mechanisms, and other imposed procedures, be supportive and lead to designs within the constraints. This is easier to state than to do in many environments.

The key to success for any design team lies in a solid understanding of the requirements and awareness of alternatives. Therefore, the methodology should capture the

requirements, highlight alternatives, foster synergy, promote and nurture creativity, support individual and team efforts, be manageable, and be comprehensible to the average intellect within the design team. The last requirement is often overlooked, yet there are limits to human intellects, and esoteric excursions into abstractions are almost always doomed to failure, regardless of their other merits.

Design Strategies

Before exploring the different strategies, tactics, and mechanisms, it's worth noting that there's significant debate over the sole use of any one group of them. The general agreement appears to be that a thoughtful combination of all these elements is likely what leads to successful designs. In this section, we will examine strategies in greater depth. Top-down design strategies are commonly recommended, and tasks are carried out in a sequence that progressively defines the layers of the system. It's a fundamental principle in the top-down method that there's a hierarchical relationship between these layers that can be identified [33].

The system design incorporates horizontal logical partitions that naturally create modularity. To further enhance this modularity, vertical partitions can be introduced within each layer. It should be noted that this is a strategic approach, and the specific tactics or mechanisms for achieving these benefits still need to be determined.

Deviations from a strict interpretation of this design procedure often occur, even in strictly software projects. For example, the target language for implementation may be fixed. This constraint may force subtle biases into the process because of inherent language limitations that influence the software architecture. Financial constraints, for example, may force the use of existing software modules, which may not be suitable [33].

At an even more pragmatic level, it is not always obvious where the top is. It is presumed that this is somehow obvious from the user requirements and specifications. Experience tends to confirm this is a contentious issue in many situations, and even consensus does not guarantee it has been found. Generally, the steps are not independent, and strict precedence relations among successive steps are often impossible to maintain. The lack of a strict precedence relation between successive steps in a top-down design process does not invalidate the concept. It means that human intellect must accommodate the constraints and imposes the necessity of iterations between steps in the design. Before discussing the impact of such iterations, it is perhaps wise to discuss the other approaches. A different approach from the top-down strategy is the bottom-up strategy, which involves successive compositions [34].

Undoubtedly, this approach has demonstrated its success and cost-efficiency in numerous systems. As long as the final expansion of the system falls within the capabilities of the presented set of fundamental modules, this methodology can be applied with effectiveness. However, two potential challenges may arise. It can be difficult to accurately anticipate whether the intended system can be executed with precision. Secondly, estimating the associated costs can also present difficulties. Therefore, a keen understanding of feasibility becomes crucial for a successful design project that begins with a diverse range of pre-existing components.

In a broader sense, all designs are constrained by existing target components. We cannot arbitrarily specify gate delays in hardware or instruction execution time in software. A design procedure must accommodate such constraints while creating a design that realizes the original requirements. A comparison of these two approaches will be

made before considering compromises. Both approaches are strategies for bridging the gap between a set of specifications and a system. Our conclusion is that whatever features of either method prove useful should be used. The top-down approach is supposed to yield a correct solution, and it can be shown to execute the requirements [34, 35].

The major criticisms in this regard are twofold. First, for a very large system, it may not be possible, in any circumstances, to prove or even demonstrate overall correctness. This is a major problem in creating complex sensor components, and second, even if it is correct, it may not be implementable within the framework of all the constraints. This is a most serious reservation in extending a strictly top-down strategy beyond software. A response-time constraint is always present in signal processing systems. Excessive modularity, imposed by the top-down approach, often leads to inefficient execution of an algorithm. On the other hand, it is much easier to optimize a modular system that is known to work than to debug a nonworking but potentially highly efficient system.

The bottom-up approach begins with basic components known to be available and working. The system is constructed by successively combining these components, in the same sense that a carpenter combines components to build a house. The major criticism is that the resulting system may not meet the specifications because the original components were inappropriate. Further arguments suggest that critical choices made at a low level may seriously constrain or inhibit decisions that must be made at higher levels. On the other hand, as the design proceeds, there is no difficulty in proving that what has been constructed thus far works. Progress payments may depend on such demonstrations [35].

A top-down approach generally takes longer to formulate the logical structure before architectural choices are apparent. In the bottom-up approach, convergence may be

faster but dependent on many intangibles such as experience and insight. There is no absolute assurance, however, that a top-down approach will yield appropriate candidate architectures or that the choice of basic components in a bottom-up approach will yield an acceptable system.

All practical designs are constrained in some way. The important requirement is to accommodate these constraints without inhibiting the innovative talents of the designers. A hybrid approach is almost always necessary; designs often proceed from both ends and meet in the middle. Indeed, as we shall now discuss, they could start in the middle and proceed outward, or vice versa [35].

The Significant-Components Approach

In this method, the system parts that operate under the most constraints are designed first. The rest of the system can be designed and integrated using other methods.

By utilizing this approach, the designer can ensure that critical operational limitations are satisfied. Additionally, there is added assurance that crucial system functions can be carried out successfully. This method requires some knowledge about the logical structure of the system. It is suggested that this approach serves as a standard for dividing the logical functions of the system. Experienced designers can often recognize the presence of crucial components in a design right from the start, regardless of the strategy employed.[35].

Constraint-Driven Design

Constraints are commonly present in numerous design environments. The design aims to meet these limitations in such situations. Constraint-driven design is a method that aligns all design strategies with the realities of critical components and constraints.

The primary feature of this approach is the explicit acknowledgment of the stage in the design process where a constraint exerts its impact. These constraints may appear as structural biases introduced by the specifications from the top or critical components inserted from the bottom. Moreover, other constraints present in the specifications or the design environment are addressed at a suitable stage in the process.

This approach has two notable advantages. Firstly, it's more likely to converge on an acceptable solution. Secondly, the constraints are addressed at the appropriate level. This method includes several advantages claimed by other approaches. In general terms, it is executed in two sequences: from the top and the edges. From the top, the design follows a logical sequence of steps that connect the specifications to a product. From the edges, the influence of any constraints is determined for each step. [36].

The strategy described here is appealing yet it is difficult in some cases to partition the constraints and to demonstrate their specific influence. Constraints might, for example, dictate a bottom-up approach or vice versa. A fundamental commonality among all the strategies discussed earlier is their attempt to modularize the overall system design. The core idea of dividing complex systems into major system modules is intuitively attractive as it enables the overall structure to be understood as a whole while concealing the implementation specifics within well-defined modules. This is recognized by reflecting upon the similarities between access graphs, which originated in operating system design, and structure diagrams, which reflect the concepts of a high-order language. The cornerstone of both approaches is the hiding of detail within well-defined conceptual modules. The procedure for arriving at such a modularized system description, following the general guidelines of any chosen strategy, is governed by a specific design tactic [36].

Control-Driven Approaches

Control-driven design approaches are generally more widely recognized. Identifying the functions to be performed, along with their necessary precedence and control features, are a natural approach for real-time processing. Historically, the development of software systems and the design of operating systems have focused first on revealing and decomposing the functions to be performed. The required data structures are constructed and integrated into the system after the process interaction has been designed. This tendency is apparent in using access graph concepts for system design, where processes and monitors constitute the main conceptual units and the data structures are implicit. One of the basic problems of simple access graph concepts is extending them to include more explicit detail concerning data structures. A more important reason for a control-driven strategy to seem natural is our general familiarity with the basic computing concept and procedure-oriented programming methodologies [36].

In a conventional processing environment, we tend to think of processing as the ordered sequential execution of procedures acting on a data set available within the system. The natural tactic for designing such systems tends to concentrate on the definition of the procedures and their ordering and coordination. New programming methodologies and architectural concepts have been developed that concentrate more on the data involved in the processing, their protection, their manipulation, and their flow throughout the overall system. These new approaches are appropriate in our area of design. Most successful implementations of data-flow-driven designs have been in software rather than as complete systems. However, signal processing has strong data-flow characteristics as well as real-time constraints. A design procedure for such systems must accommodate

both data flow and control characteristics. In addition, experience indicates that deriving either from the other often leads to a contorted system structure. In signal processing, the existence of input/output streams of data is a common characteristic. The procedural entities required to manipulate these data are also rigorously specified and constrained in real-time performance. A hybrid approach is required. The concepts of control-drive and data-driven tactics reflect only the initial part, the tip of the iceberg of a fundamental division in system operation and control [37].

Design Mechanisms

A wide variety of mechanisms is used to support design activities of all kinds. Many of these mechanisms are graphic, or partially graphic, in nature, ranging from timing diagrams, circuit diagrams, logic diagrams, and block diagrams used for hardware design to flowcharts, bubble graphs, access graphs, and structure diagrams generally used for software design. Other design mechanisms are more textual or symbolically oriented, such as algebraic transfer languages, pseudo-code, and structured English.

A basic feature of a design mechanism is that it provides a consistent set of predefined concepts and an associated representation of these concepts, with which designs are described, represented, and communicated. The major requirement placed on a design mechanism is that it is appropriate for the design process it is supposed to support.

Two important aspects of design mechanisms, in general, are apparent. At the lower levels of detail, the design mechanisms traditionally used for hardware and software designs tend to vary considerably, reflecting the general trend to treat hardware and software design as separate technological activities [38]. At a higher level of abstraction,

common design mechanisms could be used and interpreted as either hardware or software; and the more detailed the level of description represented by a specific design mechanism, the more closely it tends to be associated with either a particular tactical approach, a specific design process, or a preconceived implementation mechanism.

Consider first the traditional association of design mechanisms with either hardware or software. If we select design mechanisms that are pre-associated with describing either hardware or software, then they influence our perspective of the implementation approach from the start. Conversely, our perspective of how we will implement a design impacts our choice of the support mechanisms we select to describe the design. This may be desirable in many design environments. Suppose selected design mechanisms bias the design toward the wrong implementation approach. In that case, it means, at best, a more difficult task to reach the correct final solution for implementation and, at worst, a complete design failure [37].

Suppose we choose sufficiently abstract mechanisms that can be interpreted as hardware or software. In that case, generally, they do not allow sufficient detail to be expressed about lower-level details, and a change to another design mechanism is required once the decision is made. It is desirable to have a design mechanism that applies equally well to hardware and software yet allows for detailed design representation in both areas.

Next, consider the association of design mechanisms with either a data-driven or a control-driven approach. We must deal with both the data flow and control aspects of the system eventually. If we must use two different mechanisms, they should be easily related. Ideally, we should be able to address both aspects with a single design mechanism.

Ideals are rarely implementable in practice. What we appear to need are appropriate design mechanisms for dealing with both hardware and software descriptions from both control-driven and data-driven tactical approaches. We demonstrate that this requirement can be met with a combination of design mechanisms, one flowing smoothly into the other. Mechanisms often become inbuilt through extended use, even when they are inadequate and inefficient. It often seems easier to work longer with inefficient tools than to learn new ones. The previous familiarity provides inertia which can be difficult to overcome. The advantage of Digital Twin in its implementation with Azure makes the simulation easier [38].

Starting Point, Scope, and Scale of Design Methodologies

The starting point of a methodology should be well-defined. This point must be unambiguously identifiable; unfortunately, it rarely is. The reason is open to controversy; however, it is proposed here that it is the content of the specifications or some less rigorous requirements statement that confuses starts. If the methodology is well defined and understood, then the execution of the algorithm might tolerate several starting points. The requirement is to have all the input information required to drive the algorithm, and herein lies the problem; specifications are seldom complete, orthogonal, or structured to provide the required information.

The scope of a methodology refers to the class of problems to which it can be successfully applied. The scale refers to its capability to handle the whole problem, from the overall systems to programs and processor design. A methodology that is well suited for designing complex software systems may not be at all suitable for projects which require

a large amount of hardware and software design. In his case, the scope of the methodology appears to be too narrow [38].

Complexity and Integrity

We have provided an overview of design methodologies above. An analysis was conducted on the different structural characteristics of a design methodology with regard to its tactics, mechanisms, and strategies.

Choosing a design methodology is a complex matter that must address issues not only in the system being designed but also in project control and management. Strictly orthogonal choices are rarely possible due to the near-universal presence of numerous constraints that influence design decisions. These constraints necessitate iterations on the design steps that distort a purely successive refinement of the successive composition approach. Perhaps more significantly, these iterations require human intervention, reducing the possibility of fully automating the design.

From this overview, we can derive several fundamental conclusions. Firstly, a rigid adherence to a single, purely orthogonal design strategy is unlikely to yield optimal results for real-world design problems. Secondly, the multileveled complexity of processing systems, coupled with the need to accommodate constraints, necessitates some form of iterative procedure within any design methodology. Thirdly, applying either a control-driven or a data-driven tactic may be required at different design stages. Lastly, mechanisms that support a variety of strategies and a hybrid tactical approach are essential. A discussion of the details of the design process must wait until more concepts are developed and related to a system description mechanism. In modular processors, the modules may perform their functions in parallel and combine their results to form a final

answer. We must have a descriptive mechanism to account for this concurrency and for the implied synchronization problems [38].

CHAPTER 4

INCORPORATING DIGITAL TWIN TECHNOLOGY INTO THE FRAMEWORK: CASE STUDY

Utilizing digital twin technology enables us to address the complex issues present in the Identity and Access Management (IAM) process we previously discussed and create a secure environment where automation and provisioning can coexist. This strategy will assist industries in enhancing their IAM process to meet higher security standards. It will enable the automation and provisioning processes to work seamlessly to identify and anticipate potential issues [39, 40].

Current Identity and Access Management Issues

Many companies struggle with managing their users and ensuring their security entitlements, which various factors can cause. However, digital twin technology offers potential solutions to address these issues.

Users must sign into a large number of systems and apps, and this number is rising. Although the federation and password repository are mostly being used in corporations, but the challenge is still persistently existing. Regulatory requirements add to the administrative, auditing, and policy-enforcement burden. The process of deactivating access may need to be expedited or made more reliable to prevent users who have left the organization from maintaining access. It's crucial to ensure consistent security for privileged accounts, such as Administrator accounts, as failure to do so can result in weak accountability and allow departed users to retain access to critical systems [41]. Over time,

users can accumulate security entitlements, which can potentially lead to fraud or other abuses. Responding to audit inquiries about who has specific access, who requested and approved it, and whether it aligns with policy can be a lengthy process. Addressing these requirements and constraints is vital in showcasing the Azure implementation in our case study. Hence, we will delve into these aspects in greater detail.

Requirements and Constraints

The user requirements form the working document that drives the preparation of requirements specifications. The logical structure of a requirement specification may be assumed to be composed of two parts, the operational description and the constraints. Each part forms a link to the remainder of the design and implementation.

The operational description portion of the specification should form a reasonable image of the original requirements, augmented by the perception of what is possible as opposed to what is ideally desirable. In the following discussion, a performance measure will be considered, which provides a method of quantifying the processing required at each step. Here are implied the nonfunctional requirements imposed on the final system [48]. These constraints fall into two broad categories for our purposes; implementation limiting constraints directly limit the choice of final components and physical architectures, for example, cost, specified technology second sourced components, specified equipment, environmental constraints, etc [38].

Structural constraints force logical partitions on the system before the implementation phase, for example, modularity, fault tolerance, maintenance features, system journaling, and reliability. It combines the performance requirements outlined in the operational description and the constraints that lead to the acceptance criteria. These generally

take the form of a specific set of operational benchmarks that must be demonstrated as the final proof of compliance with the requirements.

Quantifiable Constraints

In the specification, quantifiable constraints that stem from an examination of the system and its components are taken into account. These constraints may not always be straightforward or fully comprehensible, but they are linked to numerical values that the designer must either incorporate or provide a justification for.

Quantifiable design limitations exist, with three commonly grouped into the reliability category. Reliability refers to the likelihood of fulfilling the operational function for a set duration and is impacted by various factors, including the failure mechanism model assumed. It is identified by predicting the mean time between failures (MTBF). The study of failure models and the calculation of MTBF numbers are beyond the scope of this dissertation. However, several other factors are closely related to reliability.

Availability is defined as the percentage of time the system is operational. It is related to reliability and to the mean time to repair (MTTR) a failure, i.e.,

$$A = \frac{MTBF}{MTBF + MTTR} \times 100\%$$

This attribute is useful because it relates to and focuses many decisions in the design and implementation phases [42].

It is the relationship between MTBF and MTTR that determines the availability of the system. At one extreme, systems that recover from failure in zero time regardless of the MTBF are always available but if each failure resulted in lost data, no useful work would ever be accomplished. On the other hand, extremely reliable systems that cannot be repaired are available only until the first failure. The system failure rate is a composite

function of component reliability and the redundancy built into the system. MTTR is also a function of redundancy.

For a system to be accepted by the customer, acceptance tests must be passed, which are based on quantifiable constraints. It is crucial to understand how these constraints influence design decisions, and even more important to identify which design decisions impact these constraints. Although it can be challenging to develop general guidelines, in specific cases, each quantifiable constraint should be explicitly considered during every decision made in the design process. Digital Twin technology has addressed all of these aspects.

Nonquantifiable Constraints

In nearly every set of specifications, there exists a set of attributes that are challenging, if not impossible, to quantify. These include those which cannot be accounted for in the direct design sequence; therefore, they are analyzed or accounted for after the design is completed. The limitations imposed on a design project are often not quantifiable based on a universally accepted framework, but are instead established and acknowledged for the duration of the project.

In this category fall many attributes such as modularity, extensibility, graceful degradation, and in some cases even reliability. In general, it is difficult to accommodate all constraints in any reasonable design algorithm. Indeed, the one we shall suggest in

It is difficult to provide general guidelines in this regard; however, a specific example may help. Suppose a system is proposed that has, in addition to performance, a stringent requirement for reliability and modularity in both new functions and growth. It

is straightforward to demonstrate that a proposed design achieves the desired performance. Given this, a reliability analysis could demonstrate compliance with the reliability specification. Modularity could be demonstrated by adding new functions or other growth increments. From the list of prospects that satisfy the performance requirements, a shorter list can be prepared of candidates that satisfy all the other constraints. These constraints are thus real, although challenging to deal with directly in the initial phases of a design.

The ability to offer degraded performance in the presence of a fault implies that no component can completely inhibit performance if it fails. A single-point failure is a component that, if failed, completely stops operation. These components are the first to be subjected to extensive reliability considerations and, perhaps, for replication to achieve redundancy. With the use of IoT devices and Digital Twin, in conjunction with Machine Learning, future failure will be predicted ahead of time, reducing the downtime of the operation.

The nonquantifiable constraints will form the second attribute to be considered in an algorithm for assigning logical architectures to sensors. There are examples of these in power distribution systems [43].

Logical Design

In the final implementation of any digital processing system, including comprehensive systems like Digital Twin, there are two major physical realities: the system-wide sensors and the code representing the software executed by the programmable portions of the sensors. The final system design is a detailed definition of the sensors and the software and how they work together to perform the required processing. The systems design process involves transitioning from a statement of requirements and constraints to a final

definition of the hardware and software that meets the stated requirements within the specified constraints.

A physical description of the system is given in terms of its hardware and software components. A logical description can be given regarding the logical requirements for system data flow and control. When we first attempt to compare these descriptions at the overall system level, we usually find a complex mapping between the two. That is, the distinct high-level data flow and control aspects of the system do not usually map directly either to specific hardware or software elements in a one-to-one manner.

The logical design process is viewed as a reversal of the process of architectural analysis. The analysis begins with a physical system and breaks out the data flow and control features such that the system structure and operation can be examined. Starting with a set of requirements, the logical design process entails organizing them into a logical description of the data flow and control features. This description serves as a foundation for choosing a physical system implementation.[44].

Partitioning of System Functions

When presented with the system requirements specification, the designer must first identify the primary functional requirements of the system. The requirements should then be categorized into distinct groups based on the user's perception of system functions and the corresponding operational properties of each group. There are two main categories for these system functions: grouping them into primary functional requirements and taking into account their operational characteristics. dataflow and control functions. The specification should provide sufficient information to define all the required user-visible system features, such as levels of programmability and/or modes of operation. For

instance, the specification may require the system to operate in a turnkey or fixed-function mode at the highest level yet necessitate the system to be programmable at one or lower levels of operation..

Also included in this step is the separation of the nonfunctional requirements and constraints. That is, partition the system requirements into an initial hierarchy based on specified system interfacing and functional operation requirements if such a hierarchy is obvious while setting aside the nonfunctional requirements and constraints to consider only when they explicitly impact design decisions. For each system level, starting with the top level, define the required interface characteristics in terms of the logical system functions which describe the visible characteristics of the virtual machine. This is similar in concept to the definition of the visible part, or specification, of a package or task in a structure diagram. Indeed, as we shall discuss later, this is exactly the way to use structure diagram concepts as a mechanism to support this design approach [45].

The purpose of this first step in the logical design process is to create, from the specifications, a global perspective of how the system must be structured. This usually involves numerous sketches, at the block diagram level, of various functional partitions that might prove feasible. The process is often started by postulating the system as a black box and considering the overall system interfaces required. This is logically equivalent to defining the outer interface layer of the system onion skin diagram perspective. From the interface, one could consider all the input/output signals and devices that must be attached. It is unusual to find specifications that are devoid of such requirements, although they may be part of the design goals. However, more important at this initial stage of the design process is the identification of all the logical system interfaces required. The

best guidelines that can be offered here are to carefully examine the operational description of the intended system use. From this description, a partitioning of the various modes of operation and the associated system functions for each mode should be possible. This may be considered an initial attempt to match the system requirements to the logical system perspective. This partitioning of the specified system requirements could result in a single overall virtual machine, an obvious hierarchical partitioning into several virtual machine layers, or possibly a logical structure that has more than one distinctly partitionable virtual machine description at the outermost edge of the onion skin [45].

In general, every logical interface or physical input/output device will require some tasks to manage its affairs. Indeed, experience indicates that a basic principle of structure diagram generation is to assign a task to every input/output. These may be later combined or further partitioned, but that does not violate the principle. Thus, a basic logical partitioning can be.

The partitioning criteria are usually difficult to quantify exactly. Two opposing considerations are usually evident. Partitioning must proceed until it is evident that the major functions demanded by, or implicit in, the specifications have been met; and the granularity of the partition should not proceed to such a level as to constrain the consideration of optimal solutions.

The point here is that the boundary is a gray area, and, once again, explicit directions are not forthcoming except in the form of general principles. Whenever possible, the partitioning should be organized as a hierarchical structure such that distinct levels of modularity are maintained. These levels should be matched to distinct logical levels of system interfacing and modes of operation, if possible. We note, however, that this initial

system partitioning is primarily concerned with user-visible system operation and interaction. Thus, it often concentrates on system control features, and a control-oriented approach seems applicable initially, followed by data-flow considerations later. The following general principles are offered as guidance in this first step of the logical design.

Assemble a list of the major system functions specifically required, or implied, by the specification. This list should represent all the user-visible system features that will form the basis for validating the design and establishing system acceptance criteria. Also, separate all nonfunctional requirements and constraints that are specified. Try several initial partitions of the system functions based on separable modes of system operation, noticeable logical levels of user visibility or programmability, etc [44]. The inputs to this activity will come mostly from the operational description portion of the specification if one exists, or it may be necessary to extract an operational description of the system from the performance characteristics portion of the specification. This activity will also be influenced, to a considerable extent, by the designer's perception of who will use the system and in what operational context. Stop the partitioning of a candidate as soon as a clear picture emerges that the essence of the specifications has been captured.

Several potentially useful partitions may emerge, especially if the design team has multiple designers. Provided each seems reasonable, they should all be retained, for it is rarely possible at this stage to choose an optimal solution. The next step is to design the logical architecture and refine the definition of each virtual machine, such as Digital Twin.

Logical Architectures and Virtual Machines

The second step of the logical design phase involves the creation of a logical architecture description for the virtual machine(s) defined by the system interfaces. Following either a control-driven or a data-driven tactic and using a specific descriptive mechanism, we must define the logical structure, organization, and interaction of the system functions to provide the necessary operational features of the interface. This implies a definition of the logical operation of the virtual machine, as seen from the interface above, and a description of the operation, organization, and interaction of the data flow and control functions that implement the logical interface. This may be compared to the description of the body of a package or task whose specification represents a virtual machine interface.

The logical architecture can be described using a variety of descriptive mechanisms, such as an access graph or a structure diagram. We shall use structure diagrams exclusively in this book as the final descriptive mechanism for logical architectures, although we shall also use data-flow graphs as an intermediate design mechanism which is useful in the derivation of structure diagrams. In this step, the system functions must be incorporated into alternative proposals for tasks and packages, before considering their interconnection through either rendezvous or simple calls [46]. Data-flow diagrams may be considered overly simplified structure diagrams in which the distinction between packages and tasks has been removed, and all the access arrows are omitted, leaving only nodes representing functions and the data-flow arrows. Data-flow diagrams allow a quick analysis of several possible functional decompositions of the system without having to

consider the detailed control issues of function interactions. Thus, data-flow diagrams explicitly support a data-driven tactical approach to logical systems design.

From the logical architecture description, define a set of support functions needed to implement that logical architecture. This essentially represents a decomposition of the functions of the logical architecture into a set of sub-functions for implementation. These sub-functions define the requirements of the next lower-level virtual machine for which the entire logical design cycle can be reiterated. This recursive process stops when all user-visible system levels, identified from the specification, have been defined and/or when a direct hardware implementation of the virtual machine functions seems possible within the constraints.

A design team on familiar ground can often merge these two perspectives. However, when confronted with new systems, the explicit separation of these perspectives is necessary to expose the range of possible solutions and to encourage innovations in the process. Our basic strategy is to maintain this separation and create a logical description of what the system must do before considering specific implementation possibilities. During the following discussion, we will deal only with the logical perspective [45].

The ideal scenario is that the implementation specifics should not influence the design. However, past experiences often provide valuable insights on how to proceed. There are instances where starting with a bottom-up design can be highly successful. Generally, these potential solutions should be identified, and other options should be explored before making a final decision. It's important to avoid committing to a specific hardware or software implementation. The factors that determine the best balance between these implementation methods are constantly evolving.

Regarding logical complexity, it's preferable to maintain simplicity at this stage. If a logical description of functions is excessively complex, it can lead to three issues. Firstly, the complexity is likely to increase during the implementation phase. Secondly, it could suggest that the operational requirements needed to be fully understood. Lastly, it could prevent a thorough evaluation and validation [46].

Several basic issues and alternatives must be considered in formulating logical architectures. These include not only the data-flow characteristics that must be supported but also the basic system operation and control philosophy to be followed, the specific control mechanisms required, and the issues of centralization and distribution of functions that impact scheduling and synchronization schemes. The logical architecture serves two additional functions. First, it creates the requirements for the next lower-level virtual machine to support it, and second, it becomes the object for partitioning and allocation onto hardware [47].

Partitioning and Allocation to Hardware Sensor Systems

The purpose of the logical design procedure is to create a system description that outlines what it does and how it is logically structured. Using a descriptive mechanism akin to a high-level language to define various system functions and sub-functions should not be mistaken for the final implementation mechanisms. The logical system description simply identifies what the system needs to do. Once this description is complete, it must be divided and allocated to a specific hardware architecture. It is at this stage in the design process that the full impact of both the performance requirements and the implementation constraints are felt.

The hierarchical system description could be created down to an arbitrary level of detail, such as defining the details of individual words or even bit manipulations. However, we anticipate allocating at least some system functions to hardware at a very high level when IoT sensor implementation technology is used. Thus, the allocation of major system functions to an overall system hardware architecture may take place at or near the highest-level virtual machine description. In this sense, the initial iteration of the logical design phase of this methodology is essentially the same as the basic edges-in approach. There are two major activities closely linked at this point in the design process. First, the logical system description must be partitioned for allocation to hardware, and second, an appropriate hardware architecture must be selected. The overall process of partitioning the logical architecture and allocating this partition to hardware requires that both partitioning and hardware selection be done together and iteratively [47].

One of two opposing approaches might be taken in this activity. The first approach could be to partition the logical architecture according to certain partitioning rules and guidelines. The results of various partitioning attempts could be used to select a suitable hardware architecture. The opposing approach would allow some general characteristics of known hardware architectures, or constraints, to drive the system partitioning. In practice, some combination of the two approaches is required to achieve suitable partitioning and allocation.

The known performance characteristics of various hardware architectures can be used to help develop rules and guidelines for system partitioning. The results of partitioning the system concerning such rules can suggest appropriate allocations to specific hardware structures. Considering the performance requirements and constraints, the result is

an iterative procedure whereby the alternative logical architecture partitions are allocated to various candidate hardware architectures. The resulting systems are analyzed and compared concerning performance and compliance with the specifications [42].

Partitioning

The structure diagram representation of the system could be implemented as software on a multi-purpose computer. If such an implementation satisfies the system specifications, it is a desirable way to proceed, for the implementation mechanisms are well understood. If this is not the case, then the logical system description must be partitioned for allocation onto an array of processors. Unfortunately, formal partitioning algorithms are not pragmatically viable or are limited in scope. This state of affairs prevails because of the numerous conflicting requirements on the partitions. To begin with, the partitions tend to determine the structure of the physical interconnection topologies. The partitioned algorithm may not yield a correspondence to any realizable physical architecture. Thus, an overall requirement is for a meaningful relationship to exist between logically and physically realizable partitions [48].

Partitions based on the modularity constraints require that the impact of such changes be confined to a single partition. Ideally, the addition of a new feature should be accommodated with no contamination of existing partitions. Reliability is also affected by partitioning. Failure effect analysis must ensure that failures are contained within a restricted boundary, ideally one partition or a subset of it. These failure containment boundaries can be used to detect and recover from failures. Such an analysis should also point

to the optimal components to be replicated to obtain redundancy to meet availability requirements. Finally, the partitions should enhance the ability to reconfigure the system for graceful degradation in the event of failure [43].

The problem of partitioning and allocation of the logical system description to a specific hardware/software implementation may also be approached hierarchically. The overall system, as represented by a high-level virtual machine description, may be partitioned, and allocated to a general high-level hardware architecture. The specific performance requirements that this allocation places on the various elements of the hardware may then be taken as the input virtual machine requirements to define a set of modularized subsystem design problems, which can each be approached separately by an iteration of the entire logical design and hardware selection phases. We note, however, that extreme care must be exercised in breaking the design problem down in this way. It may turn out that one or more of the subsystem design problems does not have a viable solution that meets the performance requirements and the implementation constraints. This can lead to the costly situation of a complete system redesign. Partitions are, in the end, linked closely to available hardware architectures.

CHAPTER 5

DIGITAL TWIN SIMULATION WITH CONSTRAINTS

Based on how this service is used in the real world, there are different definitions for digital twin. This section concentrates on summarizing the most often-used definition. A digital twin can be a complex machinery such as a power turbine or a simpler entity such as a water pump [49].

Digital Twin is a real-time counterpart of the running IAM process and will simulate a real IAM workflow, it uses real-time data from parts of an IAM workflow and simulates them, to analyze data from IAM process and make intelligent decisions. Digital Twin will learn and understand, how an IAM workflow is operating now, and predict how it will operate in the future for taking preventing measures or alert humans to act in timely manner. One of the primary themes is the development of digital simulation models for physical and technical objects. These models are not static; they evolve in sync with the lifecycle of the physical object. Such dynamically updated models can continuously receive new data from sensors attached to the technical object, control devices, the environment, and larger systems. They can generalize newly obtained and past data and adjust the model's parameters. Essentially, a dynamically updated model of the object is its digital twin.

The digital twin is regularly trained with fresh and historical data. It adjusts parameters in real-time, adapts to external conditions, and calculates the ideal behavior of the technological object in the current environment. Dynamic digital models allow us to

project technical objects into the digital realm, enabling us to use them to enhance the operation of technical objects, automate technical condition monitoring, diagnose faults, and predict technical conditions [49].

Digital Twin and Identity and Access Management

In IAM, the proliferation of digitalization and the complexity of connectivity demands a mechanism that can assess the operation and security of vital infrastructures. Digital Twin (DT) is changing IAM in this area. DTs are virtual replicas of real systems that mimic every aspect of a product or process and can deliver actionable insights through monitoring, optimization, and prediction, all powered by asset centric data. Furthermore, DT with replication and simulation models can prevent and discover security problems in the IAM without interfering with the live system's operations.

However, such DT advantages are predicated on a belief in data trust, integrity, and security. When it comes to the integration and interoperability of many components or sub-components among distinct DT owned by multiple stakeholders to provide an aggregated view of the complex physical system, data trustworthiness is increasingly crucial. Furthermore, processing large amounts of data in real-time to generate actionable insights is a vital requirement that necessitates automation. Finally, we identify roadblocks to adopting intelligence-driven architectures in IAM [50, 51].

Implementation Details using Azure

The procedure described here is a complete instruction to create digital twin in Microsoft Azure for this project. We will build a digital twin model for IAM processes. In this model, we will simulate users' lifecycle process from the time is onboarded to the time they are terminated.

We are going to create the Digital Twin instance, then we will create an Internet of Things (IoT) hub. We will simulate some selected sensors transmitting data from selected devices as a step in the IAM process. We will have that data ingested into the IoT hub, and then we will load that data into Digital Twin using an Azure function.

In these sections, we will demonstrate creating and setting up the environment in Azure Digital Twins [52].

Creating An Azure Digital Twins Instance

In the first step, we will create a new instance of Azure Digital Twins by logging on to the Azure portal at <https://portal.azure.com> and locate Azure Digital Twins in the Azure Marketplace [53] as depicted in Figure 12.

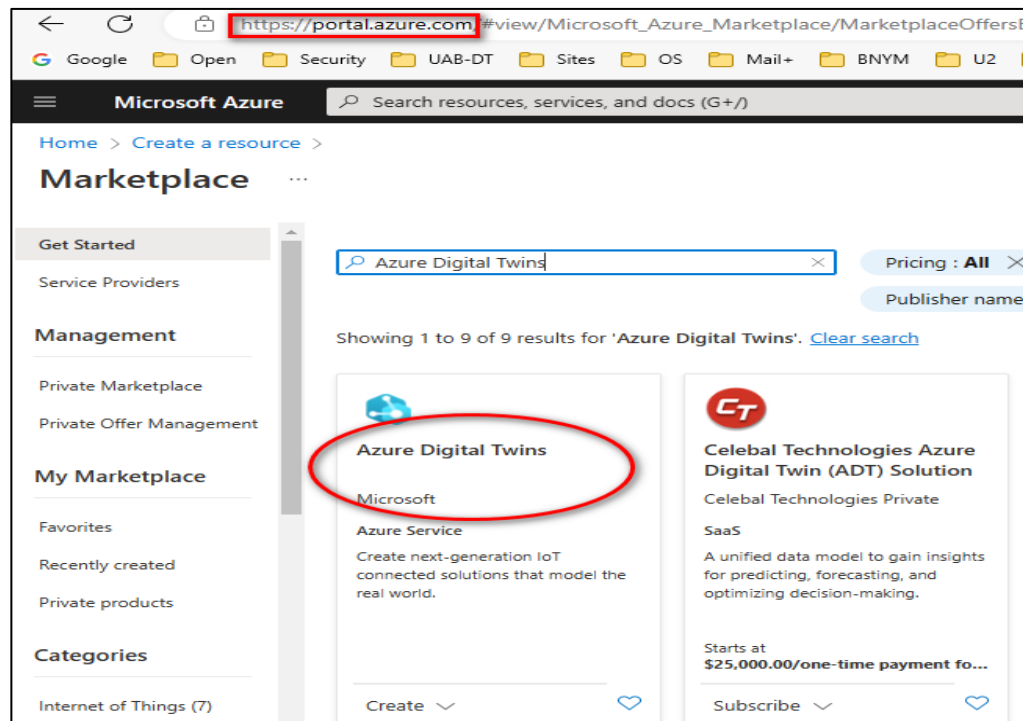
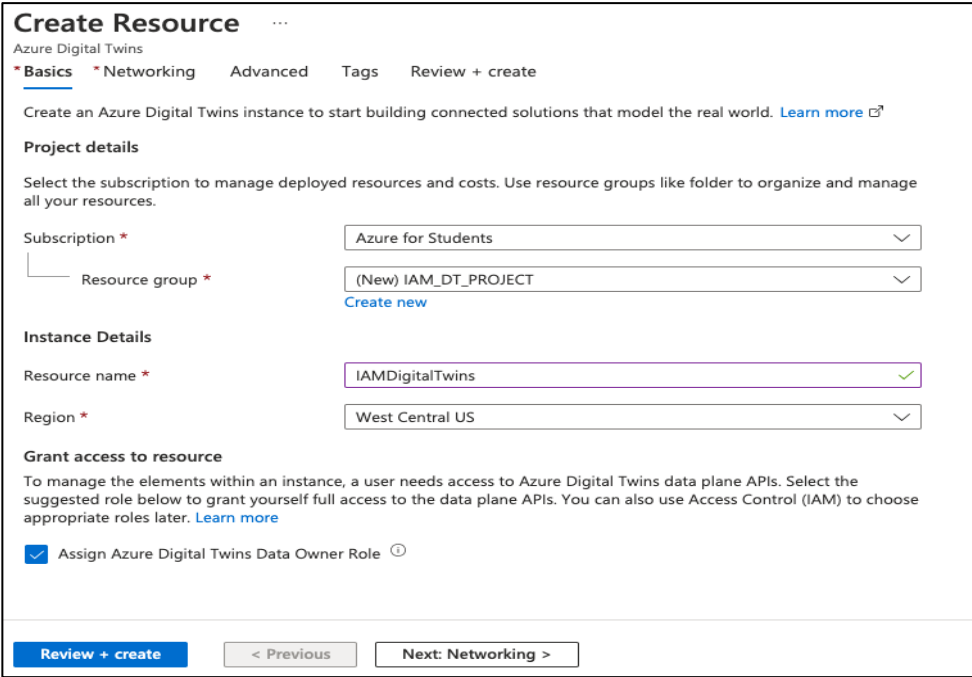


Figure 12: Locating Azure Digital Twins in Azure Marketplace.

Creating Resources

Resources are the components in Azure Digital Twins that are necessary to build an environment to simulate a real-world scenario. Azure resources will include our subscription to Azure, resource group, and geographical region. Figure 13 illustrates the creation of ADT resources. In the basic tab option, we specify the subscription to the ADT instance and the resource group associated with it. A resource name must be given along with the geographical location. We have chosen West Central US to be close to UAB where the demonstration and ingestion are performed.



Create Resource ...

Azure Digital Twins

* Basics * Networking Advanced Tags Review + create

Create an Azure Digital Twins instance to start building connected solutions that model the real world. [Learn more](#) ⓘ

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folder to organize and manage all your resources.

Subscription * Azure for Students

Resource group * (New) IAM_DT_PROJECT [Create new](#)

Instance Details

Resource name * IAMDigitalTwins ✓

Region * West Central US

Grant access to resource

To manage the elements within an instance, a user needs access to Azure Digital Twins data plane APIs. Select the suggested role below to grant yourself full access to the data plane APIs. You can also use Access Control (IAM) to choose appropriate roles later. [Learn more](#)

☒ Assign Azure Digital Twins Data Owner Role ⓘ

[Review + create](#) < Previous Next: Networking >

Figure 13: Creating Azure Digital Twins resource.

It is essential that extra attention should be given to Azure access control and roles for the resources that we build to eliminate any access issues, as it is depicted in Figure 14. Check access is chosen to view the owner access and all other delegated access. In this view, we added roles and assignments.

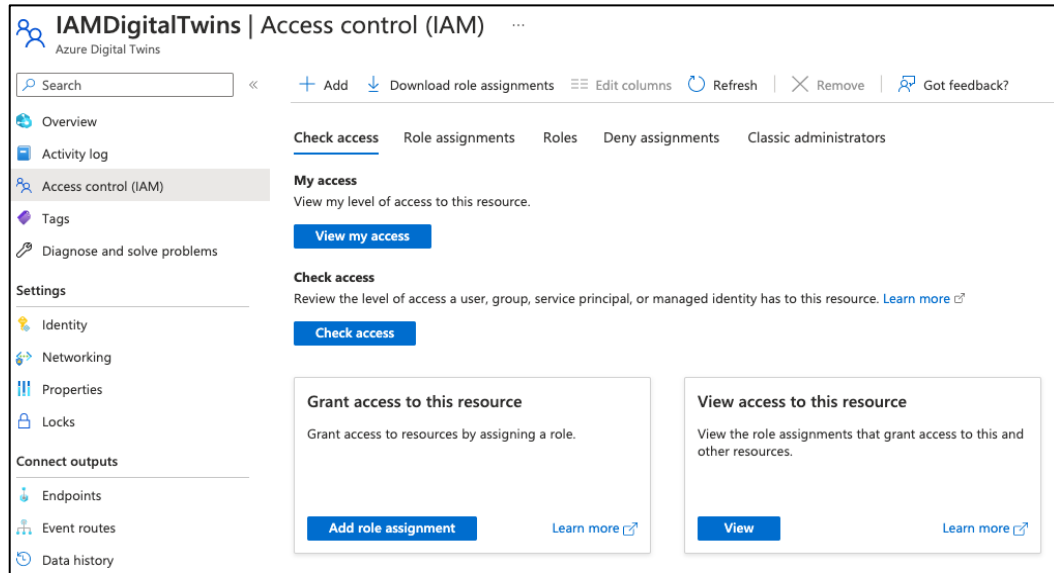


Figure 14: Azure Access control.

Figure 15 depicted the list of current access control given to users to utilize the ADT subscription. The access control within Azure is very granular which we leverage to provide access to users who are involved in this project. Azure Data Owners must be explicitly entered in the Azure AIM and given proper roles to be able to access and manipulate the data.

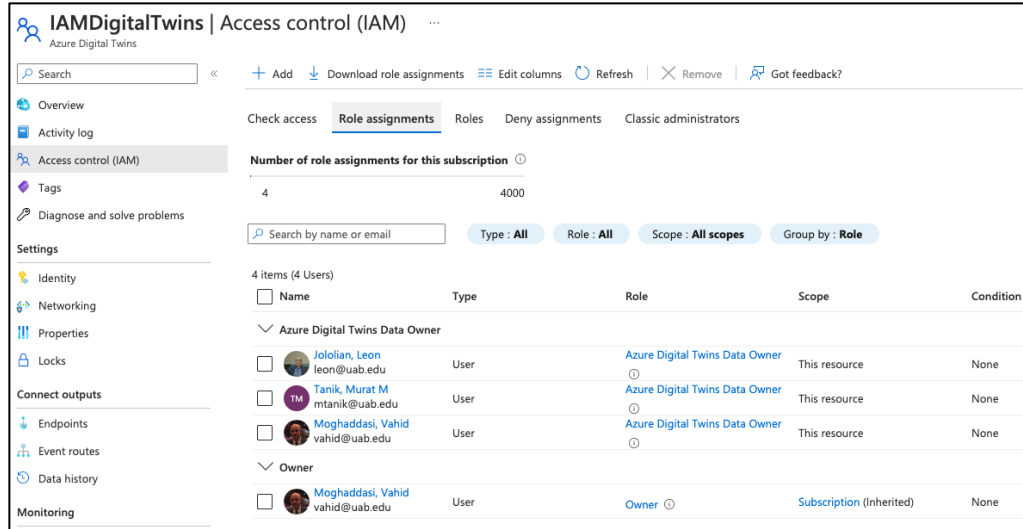


Figure 15: List and assign roles in Azure Access control panel.

Creating The Digital Twin Models

A digital twin is a virtual representation of a process or an object which simulates the lifecycle of the entity which is being used [54].

The following JSON code [55] in Figure 16, illustrates the Azure function for ingesting data from IAM devices into the IoT hub of the IAM process models. This file contains the basic tags necessary for this project, the tags are “context,” which is the metadata for the DTML and DTDL information, and the version number, in this case, version 2. “id” is the DTMI string for the type of information we have added here which is “user.” The JSON code also includes contents, consisting of type, name, schema, description, and if the data is read-only or read-write.

```

{
  "@context": "dtmi:dtdl:context;2",
  "@id": "dtmi:com:uabiamprocess:uabiamstage:entitlements:stage:user;1",
  "@type": "Interface",
  "displayName": "User – Interface Model",
  "contents": [
    {
      "@type": "Property",
      "name": "id",
      "schema": "string",
      "description": "User entitlements Id",
      "writable": true
    },
    {
      "@type": "Property",
      "name": "Name",
      "schema": "string",
      "description": "User entitlements Name",
      "writable": true
    }
  ]
}

```

Figure 16: JSON code to ingest data from sensors into devices in Azure IoT.

Digital Twin Model Identifier (DTMI) and Digital Twin Definition Language (DTDL) are defining the models and their relationship together. The keyword DTMI, and “.com” are reverse DNS [56] for UAB IAM Process entry for “UAB Identity and Access Management Process,” which is our entity as it is depicted in Figure 17.

```

    {
      "@context": "dtmi:dtddl:context;2",
      "@id": "dtmi:com:uabiamprocess:lifecycle:cycle:entitlement:user;2",
      "@type": "Interface",
      "displayName": "Authorization ",
      "contents": [
        {
          "@type": "Property",
          "name": "id",
          "schema": "string",
          "description": "User Entitlement - Validate",
          "writable": true
        },
        {
          "@type": "Property",
          "name": "access",
          "schema": "string",
          "description": "User Access - Validate",
          "writable": true
        },
        {
          "@type": "Relationship",
          "@id": "dtmi:com:uabiamprocess:lifecycle:cycle:entitlement:SelfService;2",
          "name": "SelfService",
          "displayName": "Self Service",
          "target": "dtmi:com:uabiamprocess:lifecycle:selfservice;2"
        }
      ]
    }
  ],
  "name": "New",
  "schema": "string"
}
],
{
  "@type": "Relationship",
  "@id": "dtmi:com:uabiamprocess:uabiamstage:entitlements:iamstep:rel_has_devices;1",
  "name": "rel_has_devices",
  "displayName": "Has devices",
  "target": "dtmi:com:uabiamprocess:uabiamstage:entitlements:iamstep:iamstep_device;1"
}
]
}

```

Figure 17: Relationships between Digital Twin models.

Validating the DTDL Models

We must validate the Digital Twin Definition Language that we have developed by using DTDL Validator utility which is provided by Microsoft. The utility must be compiled on the local platform. In Figure 18, you can see a successful build of the utility on MacOS.

```
% dotnet build
MSBuild version 17.3.2+561848881 for .NET
Determining projects to restore...
Restored /Users/admin/DTDL-Validator-master/DTDLValidator-
Sample/DTDLValidator/DTDLValidator.csproj (in 1.47 sec).
DTDLValidator -> /Users/admin/DTDL-Validator-master/DTDLValidator-
Sample/DTDLValidator/bin/Debug/netcoreapp3.1/DTDLValidator.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:07.77
%
```

Figure 18: Compiling DTDL Validator using dotnet utility.

We will need to run the DTDL validator against the model that we created for our project. Figure 19 illustrates a successful validation. In here, we used Microsoft utility “dotnet” to validate all the JSON code that we have developed for this project.

```
% dotnet run --directory /Users/admin/UAB_AIM_DigitalTwinsDemo-main/UABIAMModels
Simple DTDL Validator (dtdl parser library version 6.1.0.0)
Validating *.json files in folder '/Users/admin/UABIAMModels'.
Recursive is set to False

Read 5 files from specified directory
Validated JSON for all files - now validating DTDL

*****
** Validated all files - Your DTDL is valid **
*****
Found a total of 30 entities
%
```

Figure 19: Successful validation of models.

Azure Digital Twins Explorer

Azure Digital Twins Explorer is a tool that allows us to view the digital twin configuration and formatio, in Azure digital twin cloud space. This tool comes in many different forms, locally installed on your device, install in a Ducker container, Command Line Interface (CLI) [57] or web User Interface (UI). In this project, we will be using the web UI interface only. Figure 20 depicted the viewing of the URL to the current project in Azure Digital Twins. The name field is a free-hand entry by following the rules and

format of the text which we are entering. The value is the URL where the Digital Twin is deployed in Azure cloud.

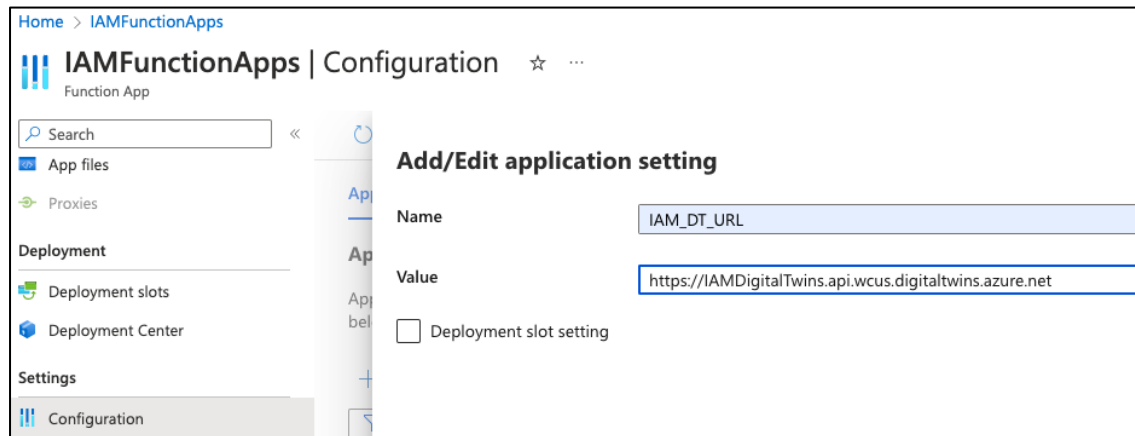


Figure 20: Retrieving Azure Digital Twins instance URL.

In Figure 21, the URL to ADT instance is loaded into the ADT explorer. The explorer needs this URL to connect to the digital twin instance which is running on Azure cloud. It is essential that we have payed attention that the security in Azure instance allows such a connection; otherwise, the explorer will fail to connect.

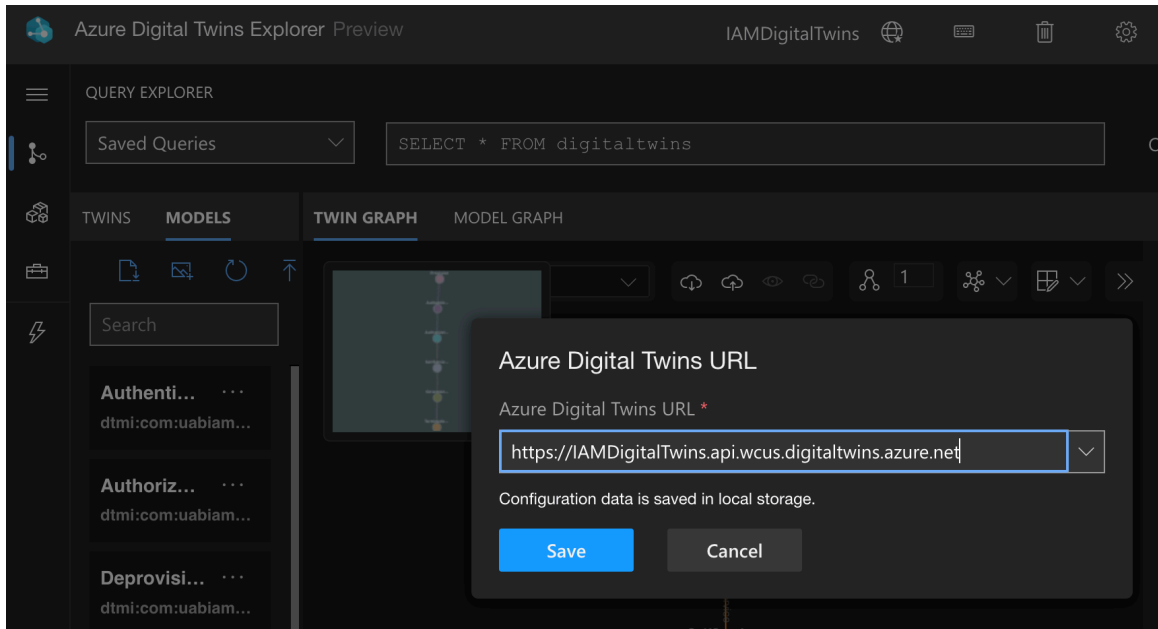


Figure 21: Entering Azure Digital Twins URL in Explorer.

Figure 22 illustrates that the ADT explorer has successfully connected to the project's ATD instance and extracted the twin from the database. As it is shown in the figure, all stages of the IAM has a relationship to another model.

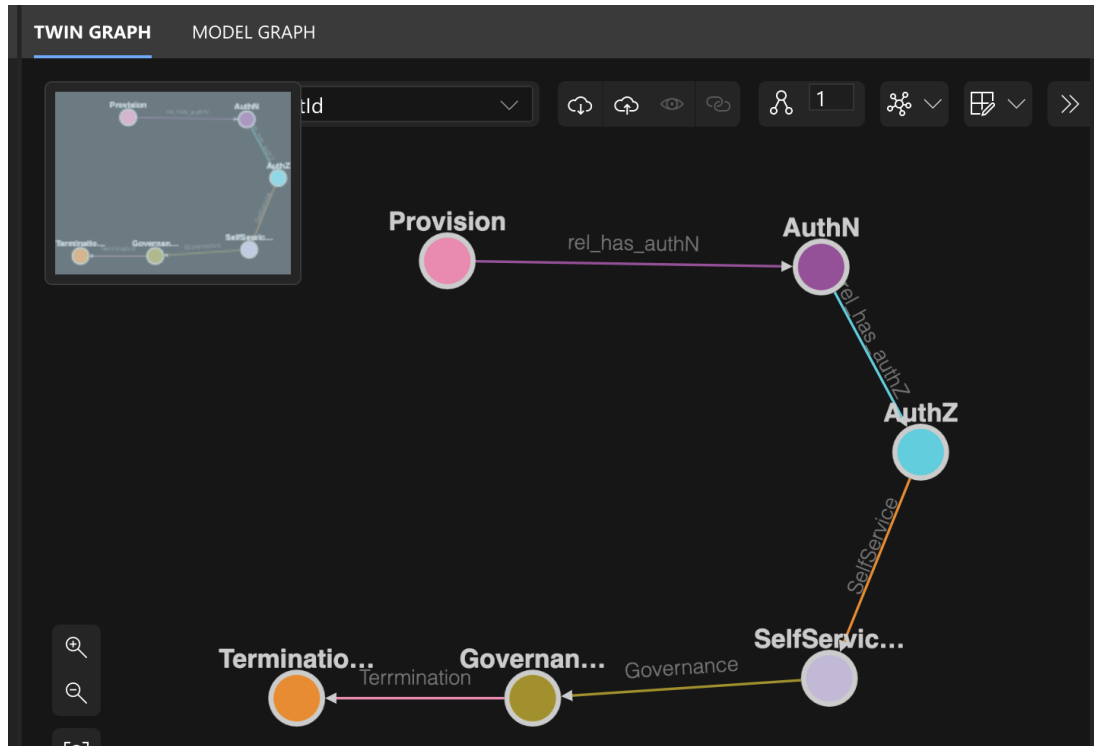


Figure 22: Azure Digital Twins explorer showing the twin.

This is one lifecycle for one user entitlement, and one system and device; in a large environment, we would have hundreds or thousands of relationships. If we have 10 users, we would have had 10 different lifecycles, and we would want to be able to see all of those, especially if the main concern is just to see immediately if one of my users has a system that is throwing a violation alert.

Importing Data from Spreadsheet

In large environments such UAB or a large-size company, we can import the relationship in bulk from a spreadsheet. This spreadsheet is specifically formatted with the DTMI relationship model IDs and the ID for specific instances- so you would fill this in with data from your database or wherever your entitlement and users are stored. Table 8 depicts CSV of the DTMI of the models for a bulk load to the ADT database, in this

file, there are Model ID to identify the model, the unique ID of the entitlement, the relationship to other models, and the initial default data.

Table 8

CSV format of DTMI for bulk load into Digital Twin

ModelID	ID (must be unique)	Relationship (From)	Relationship Name	Init Data
dtmi:com:UABiamprocess:lifecycle:cycle:entitlement;1	entitlement_504504	cycle_1234	rel_has_entitlements	{ "id": "504504" }
dtmi:com:UABiamprocess:lifecycle:cycle;1	cycle_2345			{ "cycleId": 1234 }
dtmi:com:UABiamprocess:lifecycle:cycle:entitlement;1	entitlement_253641	cycle_2345	rel_has_entitlements	{ "id": "253641" }
dtmi:com:UABiamprocess:lifecycle:cycle:system:system_device;1	system_device_63491101	system_634911	rel_has_devices	{ "id": "63491101", "violation": 12.5, "violationAlert": true }
dtmi:com:UABiamprocess:lifecycle:cycle:entitlement:user;1	user_999999	entitlement_504504	rel_has_users	{ "id": "999999", "Name": "Vahid Moghaddasi" }
dtmi:com:UABiamprocess:lifecycle:cycle:entitlement:user;1	user_605047	entitlement_253641	rel_has_users	{ "id": "605047", "Name": "Leon Jololian" }

Importing Data from JavaScript Object Notation File

There is another way to import bulk data into ADT and that is via a JavaScript Object Notation (JSON) file which has all the necessary information embedded in it. As Figure 23 depicts, in the JSON file, which is being used to upload bulk data, there are “digitalTwinsFileInfo” tag that includes the version number, user ID, user name and a unique identifier. This file also includes the DTMI information about the current model in which this file is representing for.


```

    "digitalTwinsModels": [
      {
        "@context": [
          "dtmi:dtdl:context;2"
        ],
        "@id": "dtmi:com:uabiamprocess:digitalCycle:Cycle:Entitlement:User;1",
        "@type": "Interface",
        "displayName": "User - Interface Model",
        "contents": [
          {
            "@type": "Property",
            "name": "id",
            "schema": "string",
            "description": "Entitlement User Id",
            "writable": true
          },
          {
            "@type": "Property",
            "name": "Name",
            "schema": "string",
            "description": "Entitlement User Name",
            "writable": true
          }
        ]
      }
    ]
  }

```

Figure 23: JSON code to upload bulk data into ADT.

Use The Command Line Interface to Create New Digital Twin

As mentioned, there is also a way to build the Azure IoT hub and ADT. Azure has an extensive Command Line Interface (CLI) to interact with your project. With Azure CLI, it is possible to automate all the processes and alerts as well as send the events to ADT IoT hub. Most likely, a large enterprise will have to use Azure CLI to feed the data to ADT. In our model, we have used the portal UI and refrain from many other methods which Azure provides us.

To use ADT CLI, we will need to compile the CLI tool on the platform on which our project is running.

Figure 24 illustrates the command to check the version of the CLI on a local computer. All Azure CLI commands start with 'az.' In the following output, we observed the version of the CLI, the version of the CLI core engine, and the version of the

telemetry library. There are dependencies to Azure CLI which must meet to properly work, in this case, Microsoft Authentication Library (MSAL) and Azure Management Resource libraries.

```
% az --version
azure-cli                2.46.0
core                    2.46.0
telemetry                1.0.8

Dependencies:
msal                    1.20.0
azure-mgmt-resource     21.1.0b1
Legal docs and information: aka.ms/AzureCliLegal
Your CLI is up-to-date.
```

Figure 24: Azure Digital Twins CLI.

Establishing Relationship Between IoT Digital Twin Models

The models which are written for this project must contain a workable relationship between the other models based on their behavior and function. For example, authentication should have a relationship to authorization. All the relations are defined in DTDL and inside the model files.

Figure 25 illustrates the graphical representation of ADT models and their relationship with one another. In this model, we start the relationship from Provisioning to Authentication then to Authorization and Self-Service, and from there to Governance and finally to Deprovisioning. This is a lifecycle of the IAM workflow.

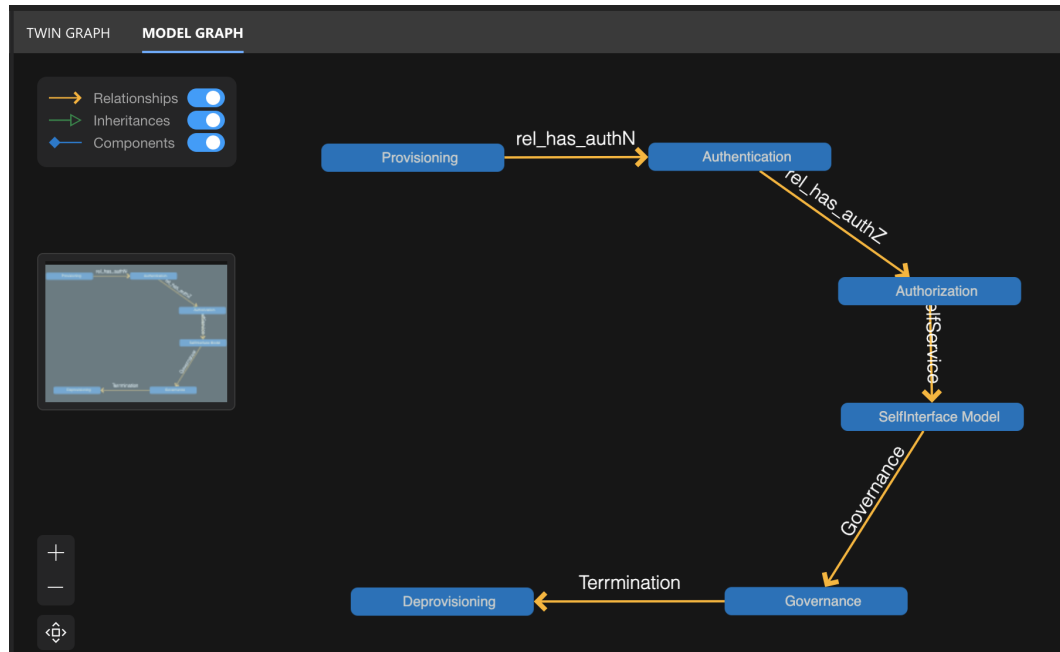


Figure 25: Model graph with established relationship.

Creating an IoT Hub

Azure IoT Hub [58] is a cloud-based solution that lets you communicate with various IoT devices and components (sensors). This is the facility we will use to communicate between our devices and our digital twin. Each of your devices will need to authenticate with the IoT hub individually so that security is ensured.

From Azure Service, we add IoT service to this project. Next, select we select the subscription and resource group which we have already created. Choose a unique name for the IoT hub and then pick a region near the location where the project is being implemented. The daily message limits differentiate each tier. You can cycle through the other options or click “Review + Create” using the default options. Once your hub is provisioned, you will find various summary statistics on the overview page for that hub. Figures 26 illustrate creating an IoT hub that will serve in this project for us as a

connector to the devices that we create. We observe that the IoT hub is being created and deployed to the region in where is closest to the project location.

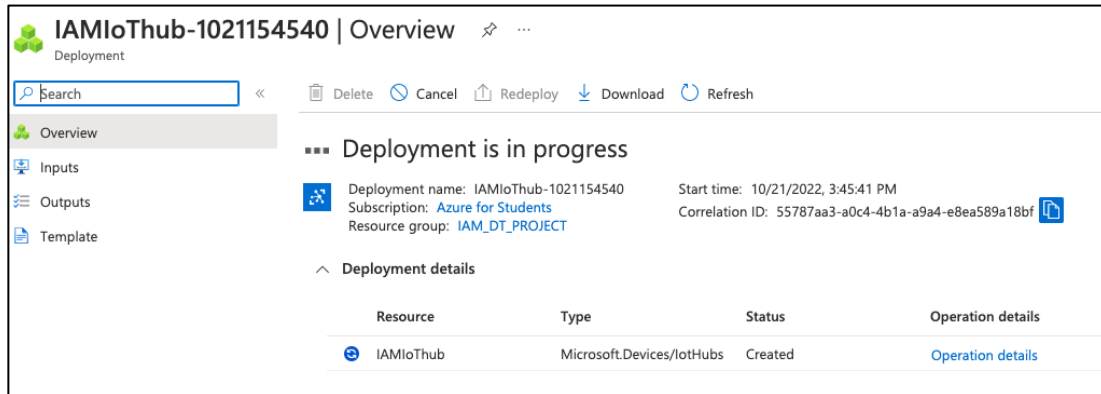


Figure 26: Creating IoT hub and being deployed to Azure cloud.

IoT hub has successfully been deployed in Azure cloud in the designated region as deprecated in Figure 27. The IoT hub is now deployed in the region where we decided to deploy it to and that is West central US which is closest to UAB location in Alabama.

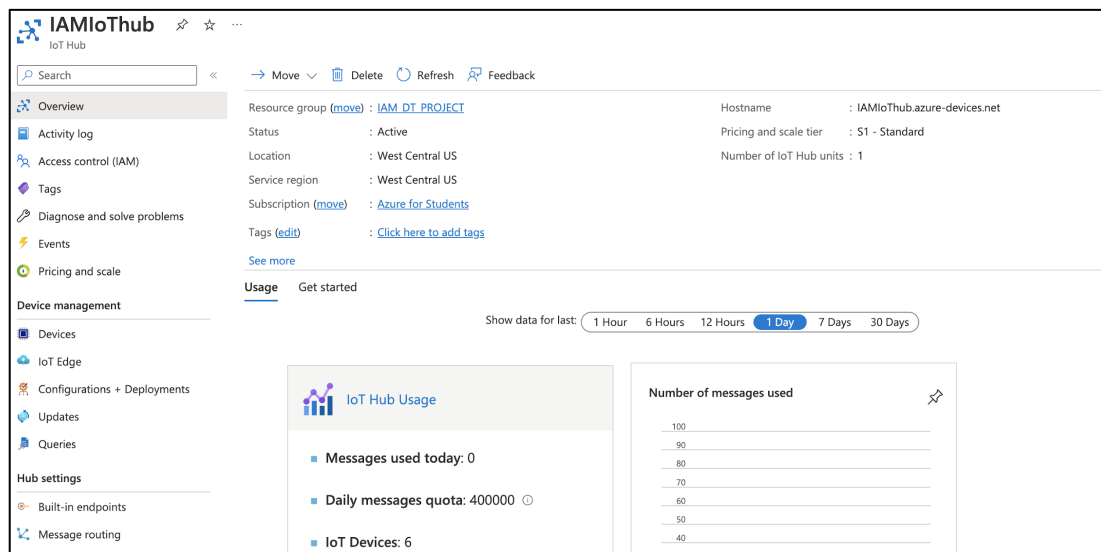




Figure 27: IoT hub is created and deployed to Azure cloud.

Adding Devices to Internet of Things Hub

At this point, we will set up the devices that we will use to communicate with the IoT Hub. From the overview page of the IoT Hub, locate the Device Management function in the left nav and then click on “Devices” to see an inventory of your configured devices. Click on “Add Device” to create a new device for your hub. We need to make sure the Device Id field matches the device id that we have defined on our digital twin. Otherwise, we will not be able to communicate with them. Next, select the appropriate Authentication type and choose “Save” to create your new device. Once the devices are created, you can test connectivity to the hub using a simulator or the actual devices via the appropriate Application Programming Interface (API) [59].

As in Figure 28 and Figure 29 depicted, we are creating devices for each process in the IAM lifecycle using the Azure ADT Explorer web UI. We have named the device unique and descriptive for simplicity and to reduce confusion. We have created all the IAM devices for this project as each device will send its own telemetry to the hub for processing. In creating devices, we include a unique device ID, Authentication type and allow the connection to this device to IoT hub.

 **Create a device** ...

 Find Certified for Azure IoT devices in the Device Catalog

Device ID * ⓘ

☐ IoT Edge Device


Authentication type ⓘ
Symmetric key X.509 Self-Signed X.509 CA Signed


Auto-generate keys ⓘ
☒

Connect this device to an IoT hub ⓘ
Enable Disable

Parent device ⓘ
No parent device
[Set a parent device](#)

Figure 28: Creating provisioning IoT device.

 **Create a device** ...

 Find Certified for Azure IoT devices in the Device Catalog

Device ID * ⓘ

☐ IoT Edge Device

Authentication type ⓘ
Symmetric key X.509 Self-Signed X.509 CA Signed

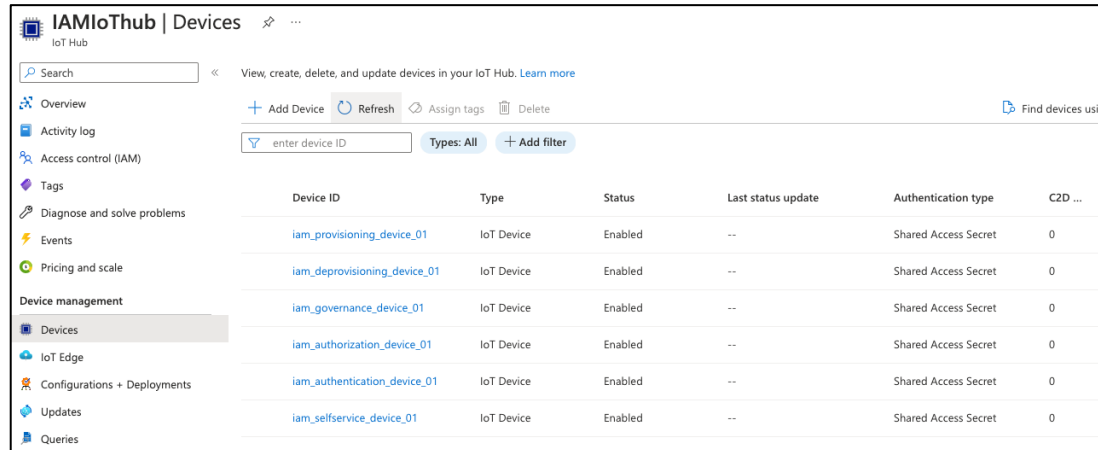
Auto-generate keys ⓘ
☒

Connect this device to an IoT hub ⓘ
Enable Disable

Parent device ⓘ
No parent device
[Set a parent device](#)

Figure 29: Creating authentication IoT device.

We must list all the devices that we created in the device list shown in Figure 30. This step is necessary to ensure that all the IoT devices have been created and functional. Any error message in this step should be taken care of before moving on to the next step.



The screenshot shows the 'IAMIoThub | Devices' page in the Azure portal. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Pricing and scale, and Device management. Under 'Device management', 'Devices' is selected. The main area shows a table of devices with columns: Device ID, Type, Status, Last status update, Authentication type, and C2D There are six devices listed, all of type 'IoT Device' and status 'Enabled'.

Device ID	Type	Status	Last status update	Authentication type	C2D ...
iam_provisioning_device_01	IoT Device	Enabled	--	Shared Access Secret	0
iam_deprovisioning_device_01	IoT Device	Enabled	--	Shared Access Secret	0
iam_governance_device_01	IoT Device	Enabled	--	Shared Access Secret	0
iam_authorization_device_01	IoT Device	Enabled	--	Shared Access Secret	0
iam_authentication_device_01	IoT Device	Enabled	--	Shared Access Secret	0
iam_selfservice_device_01	IoT Device	Enabled	--	Shared Access Secret	0

Figure 30: List of all devices that we created in IoT hub.

Once the devices are created successfully, the device string and access security key will be generated by Azure IoT and assign to each device. These keys can be regenerated if need be in case of leaking out and potential security and vulnerability issue. Figure 31 depicted the location of the connection string to the device and the secret keys to communicate with the device. We need this information to securely communicate with the device and prevent intruders to use the device.

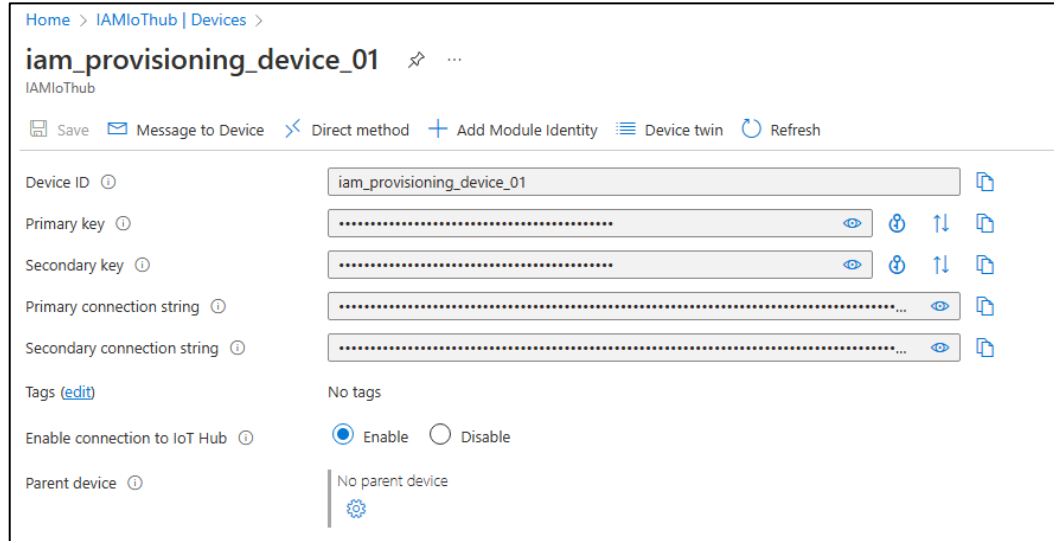


Figure 31: Device ID, Shared key, and connection string.

Creating an Azure Function App

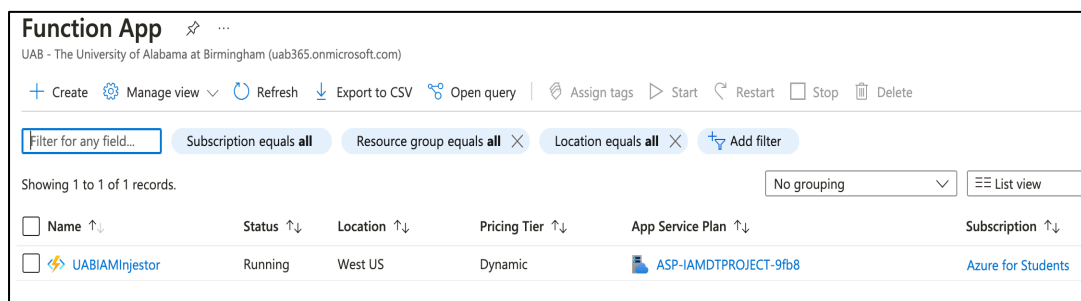
Azure Function App is a serverless computing service offered by Microsoft Azure [60]. It provides developers with the capability to develop and deploy small, independent pieces of code known as "functions" without the need to manage infrastructure [61].

Supported programming languages in Azure Function App, including C#, Java, JavaScript, Python, and PowerShell. Each Azure Function App comprises multiple functions, each functioning autonomously and triggered when specific events or conditions occur. These events can encompass handling HTTP requests, processing messages from Azure Service Bus, responding to changes in Azure Storage, or triggering based on predefined schedules.

In the context of Azure Digital Twins, the Azure Function App can be utilized to respond to data received from IoT devices and other sources. To transmit telemetry data from physical devices or a simulator to the Azure Digital Twins instance, it is necessary to create an Azure function responsible for ingesting the telemetry data published to the

IoT Hub. The function that receives the telemetry events can then update properties on the Azure Digital Twins instance or generate events within the instance. To create a Function App, we select Function App from the Azure Marketplace [53]. We select the subscription and Resource Group. We chose a unique name for the function app and selected “Code” under Publish option. Choose the appropriate Runtime stack, Version, and Region. We selected a storage account. We used Consumption (Serverless) for the Plan type. We selected the appropriate option for networking and monitoring, and finally, we “Review + Create”.

We need to create a data owner role for the function app so that it can be authenticated with Azure services. We use “Azure role assignments” to add a new role assignment and choose “Resource group” for the scope of this project we choose the appropriate resource group. We paid extra attention to make sure that everyone involved in the project has “Azure Digital Twins Data Owner” role or you risk running into problems when we try to ingest data from your IoT Hub. Figure 32 illustrates the list of Function App which we have already created for this project. The Function App name is a unique and descriptive for simplicity, we choose UABIAMInjstor.



Name	Status	Location	Pricing Tier	App Service Plan	Subscription
UABIAMInjstor	Running	West US	Dynamic	ASP-IAMDTPROJECT-9fb8	Azure for Students

Figure 32: List all Function App.

We have configured Function App to be used by this project and to have a secure connection to be made to it. Figure 33 depicts the configuration of the Function App. In IAMFunctionApps configuration, we have set environment variables pertaining to our project as the default values are unsuitable. We will need to create an environment variable for the service URL used by the function app to connect to the digital twin instance. The URL environment variable is stored in ADT_SERVICE_URL name field. The value should be the hostname of our Azure Digital Twins instance prefix by “https://”.

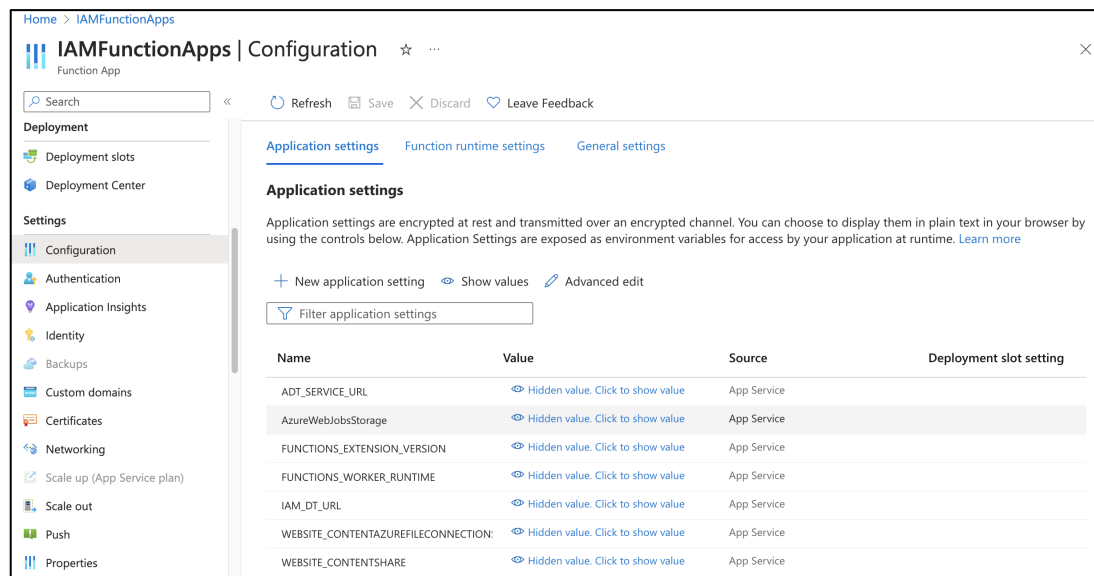


Figure 33: Function App configuration.

Building Function to Ingest the Telemetry Data

Telemetry data in Azure refers to the information collected from various Azure resources, such as virtual machines and applications, for monitoring and analysis purposes [40, 41]. We have developed the telemetry ingestion function from scratch using instructions from the documentation or use the sample code available on the

Microsoft Digital Twin Tutorial. We will load the project into Microsoft Visual Studio and build the project. We will then publish the function onto Azure directly from Visual Studio.

Event Subscription in Dealing with IAM Exceptions

Azure Event Subscription is a feature in Microsoft Azure that enables the routing of events from various Azure services to subscriber endpoints [62]. To connect events from IoT Hub to the Function App for ingesting the events, we must create an event subscription for the IoT Hub. This is done from the IoT Hub overview page, where “Event” on the left hand nav to create an event subscription. Event Schema=“Event Grid Schema”. We chose a name for “System Topic Name”. Under the event type, we checked “Device Telemetry” and unchecked everything else. Endpoint type is visible by Type=“Azure Function” and then selecting the desired endpoint matching the target resource group, Function App, slot, and lastly, the actual function.

Developing Simulator to Send Telemetry Data to IoT Hub

The Azure Digital Twins Simulator is a tool provided by Microsoft Azure for simulating and testing digital twin applications [63]. We will use a simulator to test the interoperability among our Azure Digital Twins components before bringing our devices online. There are samples available in various languages that we can use to assemble our simulator. For our project, we are using the C# version. We need to make a few modifications to the boilerplate codes to connect to our IoT Hub and send the telemetry data that matches our models. We will construct the connection information using the name and shared access key of our IoT Hub, in addition to the name and shared access

keys of our IoT devices. For the telemetry data, we must create a data structure in C# using the corresponding digital twin models.

After we have assembled the codes for our simulator, we need to compile the code using the dotnet Software Development Kit SDK [63]. Once our simulator code is compiled cleanly, we can then run the simulator from a command line interface. Figure 34 illustrates a successful compilation of the ingest code on the local computer. In this project, we used the Microsoft utility, “dotnet” version 17.6.1 to compile the source code.

```
% dotnet build
MSBuild version 17.6.1+8ffc3fe3d for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  DeviceSimulator -> /Users/admin/Library/CloudStorage/Dropbox/My Mac (admin's MacBook Air)/Documents/UAB/PHD/FINAL/UAB_IAM_DT_Demo_01/Simulator/UABIAMSimulator/DeviceSimulator/bin/Debug/netcoreapp3.1/DeviceSimulator.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.29
% █
```

Figure 34: Building the simulator.

Ultimately that is how we will send back the IAM data to ADT to ingest. Generally, what we used for this project is the hub name, shared access key, and each of the shared access keys for all the sensors that we are using, as depicted in Figure 35.

In this telemetry simulator, we are including predefined libraries such as System, System Collections, System Text, and System Threading.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DeviceSimulator
{
    public class UABIAMTelemetry
    {
        public string id { get; set; }
        public double iamprocess { get; set; }
        public bool iamprocessAlert { get; set; } = false;
    }
}
```

Figure 35: Telemetry Simulator.

End To End Simulation for IoT Functionality

We should now be able to send simulated telemetry data to the IoT Hub using our simulator, which our function app will then ingest [64]. The Azure portal provides us with facilities that allow us to manage all aspects of our IoT Hub, devices and digital twin instances. There are several visual metrics available in the IoT Hub and Function App summary pages. These will aid in verifying and troubleshooting connectivity between our simulator, IoT Hub, and our IoT Hub and Function App.

The Metrics tab on the Function App summary page has statistics such as function executing count and amount of data consumed by time. The streaming logs can be accessed via the “Log stream” [65] menu from this page. The log stream is great for confirming we are receiving data from the IoT Hub. If there are errors or exception violation, we should be able to see them in the logs.

Once we have confirmed data are flowing from the Simulator to the hub and then to the Azure Function App, we should be able to see the updates on our digital twin via the explorer. If for some reason our twin are not getting the updates, then we should

check to make sure the properties on our twin have been initialized with the appropriate credentials and shared keys. This is one of the most common problems regarding receiving updates. We summarised a typical scenario of our case study in Table 9 Case Study Scenario.

Table 9

Case Study Scenario

- It is typical within an enterprise for users to move from one department to another.
- The initial department might be a technical group providing the employee (user) access to operational-critical servers.
- On the other hand, the new department could be a managerial group.
- Both the old and the new managers (System Access Authorizer) independently have the responsibility of making sure the user has not carried over the entitlements to the new group.
- The new manager must determine the privileges needed for the new role and accordingly submit/verify/approve the new access.
- The old and new access rights are not mutually exclusive the complexity is partially in the fact that a typical employee may have hundreds if not thousands of access rights.
- Another complexity is that each entitlement for an employee is kept in a different repository. These repositories are maintained in decentralized distributed systems.
- Further complexity is introduced because the enterprise would acquire yet another entitlement system to monitor and track users' entitlement on different systems, primarily manually.
- This entitlement system will ask SAA to verify the access of the employee to hundreds or thousands of systems and resources, which often gets approved by bulk due to the amount of manual work that has to be done.

CHAPTER 6

SUMMARY, CONCLUSIONS, AND FUTURE WORK

Access to private data and manipulating that data by unauthorized users has always been a problem and challenge. It is usually government regulation and an enterprise policy to prevent unauthorized processes. Identity and Access Management (IAM) is an essential part of any enterprise, which has to be improved.

Figure 36 illustrates the summary of an IAM process model in an organization.

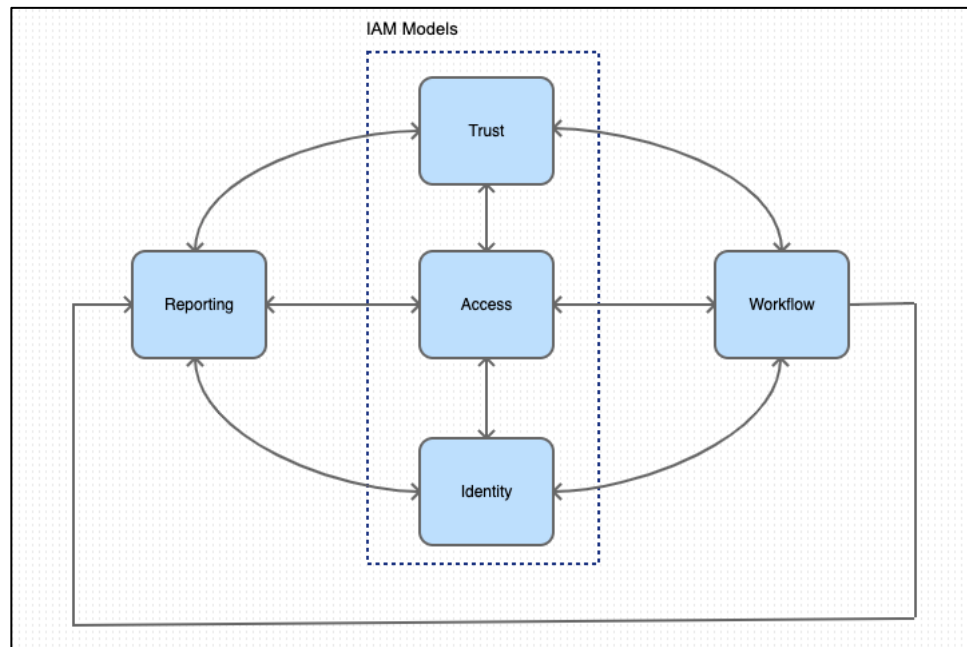


Figure 36: IAM Summary.

We are proposing to treat IAM as a formal process by which enterprises provide access to their digital assets and resources. The complexity of IAM emanates from the

rapidly changing business environment requiring constant monitoring and updating of entitlement to assets. In this dissertation, we present a dynamic framework for IAM based on the Internet of Things architecture analysis with simulation by utilizing digital twin technology.

The goal of our approach is in its responsiveness and continuous use of data feedback coming from process monitoring sensors and systems. Furthermore, our experience of an adaptive business improvement methodology used for enterprise design called Internet of Things architecture is used in our design approach. We have been working on the IAM space for the last two decades. Our current knowledge, with the associated issues, has guided our research in the design of this new framework. Our preliminary pilot identification and implementation further guided our design of this framework we named: *A Dynamic Identity and Access Management Process (DYN-IAMP)*. Furthermore, our implementation incorporates the “Drag-and-Drop Communication of Data” patent developed at UAB. Our framework, DYN-IAMP, is a new dynamic process approach through the development of core competency in the domain of IAM. Figure 1 in the introduction above faithfully depicts the proposed framework.

The implementation of Azure Digital Twins in Identity and Access Management (IAM) and the use of machine learning [64] technology presents a transformative approach to managing digital identities and access controls. Azure Digital Twins is a service that enables the creation of comprehensive digital models of an environment. When integrated with IAM, it can provide a detailed representation of users, their roles, and access privileges within a system.

We have completed Internet of Things end-to-end Simulation as shown in Figure 37. This end-to-end Simulation is implemented using Azure Digital Twins technology.

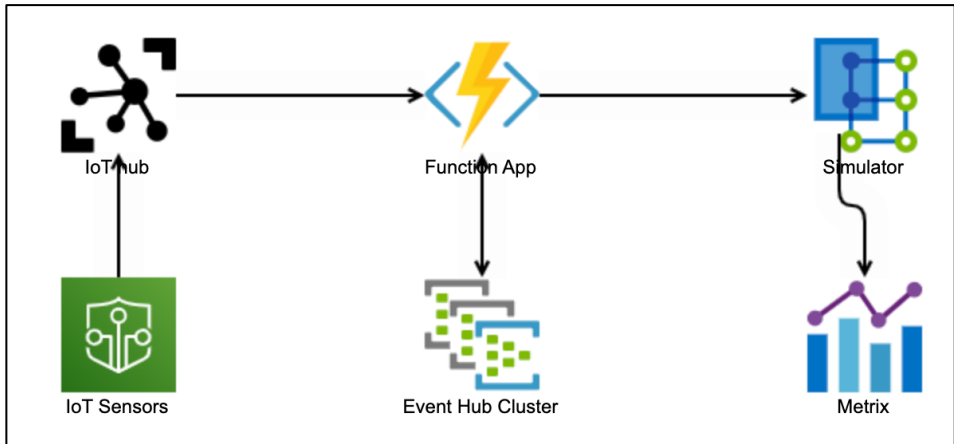


Figure 37: Internet of Things end-to-end Simulation.

The summary of the end-to-end simulation is shown in Table 10.

Table 10

Summary of applying the end-to-end Simulation

<ul style="list-style-type: none"> • User access is monitored in real time, and alerts are generated if access is altered outside the predefined parameters. • If a user moves to a new department, entitlement will change to the predefined access, which is set in the simulator engine. • Every previous access right which the user had will be revoked, and new access rights will be applied according to the new position and role.
--

The summary and conclusion of our environment are shown in Table 11.

Table 11

Summary and Conclusion

- The simulation environment serves as a valuable tool for understanding, testing, and improving access management processes in a distributed computing environment in an enterprise.
- The simulation environment can be used to analyze and optimize access management processes. It can provide insights into potential bottlenecks, security vulnerabilities, or inefficiencies in the processes.
- The proposed architecture uses a real-time simulation environment to validate and ensure that access management requirements are met.

This digital twin model can be used to simulate and analyze potential security threats, thereby enhancing the overall security posture. It can also help understand the impact of any changes to access controls before they are implemented, reducing the potential for unforeseen security issues.

The integration of machine learning [64] technology further enhances this approach. Machine learning algorithms can be used to analyze patterns in user behavior and access requests. This can help identify anomalies that might indicate potential security threats, such as unusual access requests or changes in user behavior.

Furthermore, machine learning can also be used to automate granting or revoking access privileges based on these patterns, making the IAM system more dynamic and responsive. This combination of Azure Digital Twins and machine learning technology in IAM represents a powerful tool for enhancing security and efficiency in digital systems.

LIST OF REFERENCES

- [1] Harvard University. "Identity and access management program plan." https://iam.harvard.edu/files/iam/files/iam_program_plan.pdf (accessed Nov. 9, 2021).
- [2] S. Boschert and R. Rosen, *Digital Twin—The Simulation Aspect*, Siemens AG, Corporate Technology: Munich, Germany, 2016
- [3] Microsoft, "Azure Digital Twins." [Online]. Available: <https://azure.microsoft.com/en-us/services/digital-twin/>. [Accessed: June 9, 2023].
- [4] P. Fremantle, B. Aziz, J. Kopecký, and P. Scott, "Federated Identity and Access Management for the Internet of Things," 2014. [Online]. Available: <https://dx.doi.org/10.1109/SIoT.2014.8>. [Accessed: June 3, 2021].
- [5] M. Nuss, A. Puchta, and M. Kunz, "Towards Blockchain-Based Identity and Access Management for Internet of Things in Enterprises," 2018. [Online]. Available: https://dx.doi.org/10.1007/978-3-319-98385-1_12. [Accessed: June 3, 2021].
- [6] R. T. Yeh, K. E. Pearlson, G. Kozmetsky, *Zero Time: Providing Instant Customer Value - Every Time, All the Time!*, Bridgewater, NJ: John Wiley & Sons, 2000.
- [7] M. E. Porter, *Competitive Advantage: Creating and Sustaining Superior Performance*. New York, NY: Free Press, 2008.
- [8] J. Basney, H. Flanagan, T. Fleury, J. Gaynor, S. Koranda, and B. Oshrin, "CILogon: Enabling Federated Identity and Access Management for Scientific Collaborations," [Online]. Available: <https://dx.doi.org/10.22323/1.351.0031>. [Accessed: June 3, 2021].
- [9] P. R. Carnley and H. Kettani, "Identity and Access Management for the Internet of Things," 2019. [Online]. Available: <https://dx.doi.org/10.18178/ijfcc.2019.8.4.554>. [Accessed: June 3, 2021].
- [10] M. Tanik, L. Jololian, and R. S. Sadasivam, "Drag-and-Drop communication of data via a computer network," WO Patent 2,007,008,687, Jan. 18, 2006. [Online]. Available: <https://patents.google.com/patent/WO2007008687A2/ar>

- [11] U. Tanik, M. Tanik, and L. Jololian, "Internet enterprise engineering a 'zero-time' framework based on 't-strategy'," in *Proceedings of IEEE SoutheastCon 2001*. Mar. 30-Apr. 1, 2001. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=923127>
- [12] D. Romero and F. Vernadat, "Enterprise information systems state of the art: Past, present and future trends." *Computers in Industry*, vol. 79, pp. 3-13, Mar. 2016, doi:10.1016/j.compind.2016.03.001.
- [13] A. Smith and B. Johnson, "Simulating Real-World Scenarios with Azure Digital Twin," in *International Conference on Simulation and Modeling (SIMMOD)*, pp. 1-5, 2022.
- [14] M. Grieves, *Virtually Perfect: Driving Innovative and Lean Products Through Product Lifecycle*, Merritt Island, FL: Space Coast Press, 2011.
- [15] H. Liu, X. Wang and Q. Quan, "Research on the Enterprise' Model of Information Lifecycle Management Based on Enterprise Architecture," 2009 Ninth International Conference on Hybrid Intelligent Systems, 2009, pp. 165-169, doi: 10.1109/HIS.2009.246.
- [16] R. E. Riachetti, *Design of Enterprise Systems: Theory, Methods, and Architecture*, Boca Raton, FL: CRC Press, 2010.
- [17] P. J. Hagan, *The MITRE Corporation, Enterprise Architecture Book of Knowledge*, McLean, VA: MERIT Corporation, 2004
- [18] M. Tanik and A. Ertas, *Design as a basis for unification: System Interface Engineering. Computer Applications and Design Abstraction Symposium, ASME- ETCE Conference PD-43*, pp. 113-115. Houston, TX, Jan 1992.
- [19] S. Sreerangaraju, "Emulation vs. Simulation," 29 May 2020. [Online]. Available: <https://www.perfecto.io/blog/emulation-vs-simulation>. [Accessed Sep 2021].
- [20] B. Martin, B. Hannington, *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*, Beverly, MA. Rockport Publishers, 2012.
- [21] C. A. Tavera Romero, J. H. Ortiz, O. I. Khalaf, and W. Montilla Ortega, "Software Architecture for Planning Educational Scenarios by Applying an Agile Methodology," *Int. J. Emerg. Technol. Learn.*, vol. 16, no. 08, pp. pp. 132–144, Apr. 2021.
- [22] R. W. Virden, "A Framework Platform for Object-Oriented Distributed Application Development," MS Thesis, Com Sci. Dept., SMU, Dallas, TX. 1994.
- [23] R.S. Pressman, D. Lowe, *Web Engineering: A Practitioner's Approach*, New York, NY. McGraw-Hill, 2009.

- [24] B. A. Bowen and W. R. Brown, Systems Design of VLSI Systems Design for Digital Signal Processing. Englewood Cliffs, NJ. Prentice-Hall, Inc., 1985.
- [25] G. E. Dieter, L. C. Schmidt, Engineering Design: Sixth Edition, New York, NY. McGraw-Hill, 2021.
- [26] N N. Dedić, "EAFP: Enterprise Architecture Fusion Process," *JIOS*, vol. 45, no. 1, Jun. 2021.
- [27] M. De Vries, A. Gerber, and A. van der Merwe. In: Aveiro D., Tribolet J., Gouveia D. (eds) "The Nature of the Enterprise Engineering Discipline." *Advances in Enterprise Engineering VIII*, vol. 174, pp. 1-15, May 2014, odi: https://doi.org/10.1007/978-3-319-06505-2_1
- [28] L. K. P. D. Gunawardhana, "Process of Requirement Analysis Link to Software Development," 2019. [Online]. Available: PDF. [Accessed: June 20, 2022].
- [29] C. A. Costa, and P. Baltus, (2021). Design Methodology for Industrial Internet-of-Things Wireless Systems. *IEEE Sensors Journal*. DOI: 10.1109/JSEN.2020.3031659
- [30] S. Albin, The Art of Software Architecture: Design Methods and Techniques, Indianapolis, IN. Wiley Publishing, 2003.
- [31] J. W. Ross, P. Weill, D. C. Robertson, Enterprise Architecture as Strategy: Creating a Foundation for Business Execution, Boston, MA. Harvard Business School Press, 2006.
- [32] A. G. Kleppe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Upper Saddle River, NJ. Addison-Wesley, 2009.
- [33] C. E. Otero, Software Engineering Design: Theory and Practice, Boca Ranton, FL. Taylor & Francis Group, 2012.
- [34] K. Caputo, CMM Implementation Guide: Choreographing Software Process Improvement, Reading, MA. Addison-Wesley, 2001.
- [35] F. F. Tsui and O. Karam, Essentials of Software Engineering, Burlington, MA. Jones and Bartlett Learning, 2011.
- [36] K. Hammer and T. Timmerman, Fundamentals of Software Integration, Sudbury, MA. Jones & Bartlett Learning, 2008.
- [37] W. Royce, Software Project Management: A unified framework, Boston, MA. Addison-Wesley Professional, 2000.

- [38] R. E. Fairley, *Managing and Leading Software Projects*, Hoboken, NJ. Wiley, 2009.
- [39] Microsoft. (2021, March 24). Azure Digital Twin. [Online]. Available: <https://azure.microsoft.com/en-us/services/digital-twin/>. [Accessed: March 10, 2023].
- [40] Microsoft. "Azure Monitor overview." [Online]. Available: <https://docs.microsoft.com/en-us/azure/azure-monitor/overview>. [Accessed: March 11, 2023].
- [41] V. Bolgouras, A. Angelogianni, I. Politis, and C. Xenakis, "Trusted and Secure Self-Sovereign Identity framework," in *Proceedings of the ACM*, 2022. [Online]. Available: <https://dx.doi.org/10.1145/3538969.3544436>. [Accessed: March 25, 2023].
- [42] Y. F. Li, H. Z. Huang, Y. Liu, N. C. Xiao, and H. Q. Li, "A new fault tree analysis method: fuzzy dynamic fault tree analysis," *Eksploracja i Niezawodnosc-Maintenance and Reliability*, vol. 14, no. 3, pp. 208-214, 2012.
- [43] B. Saad and A. Abene, "Rational Reliability Centered Maintenance Optimization for Power Distribution Systems," *Electrical Power and Energy Systems*, vol. 73, pp. 350-360, 2015. [Online]. Available: <https://doi.org/10.1016/j.ijepes.2015.05.015>. [Accessed: Apr 3, 2023].
- [44] M. M. Tanik, E. S. Chan, *Fundamentals of computing for software engineers*, New York, NY. Van Nostrand Reinhold, 1991.
- [45] E. M. Roche, *Managing information technology in multinational corporations*, New York, NY. Macmillan Publishing Company, 1992.
- [46] P. Watson, *Convergence: The Idea at the Heart of Science*, New York, NY. Simon & Schuster, 2017.
- [47] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, New York, NY. McGraw-Hill, 2005.
- [48] A. Shrivastava, H. Kumar, S. Kapoor, S. Kumar, and M. Balakrishnan, "Optimal hardware/software partitioning for concurrent specification using dynamic programming," in *Proceedings of the 13th International Conference on VLSI Design, Wireless and Digital Imaging in the Millennium*, Calcutta, India, 3-7 January 2000, pp. 110-113.
- [49] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.

- [50] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, "A survey on AI driven digital twin in industry 4.0: Smart manufacturing and advanced robotics," *Sensors*, vol. 21, no. 19, 2021.
- [51] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108952–108971, 2020.
- [52] Microsoft. "Create an Azure Digital Twin instance." Microsoft Docs, 2021.
<https://docs.microsoft.com/en-us/azure/digital-twin/how-to-set-up-instance>.
- [53] "Azure Marketplace," in *IEEE Std 2000-2019, The IEEE Standards Collection*, pp. 325-326, 2019
- [54] Microsoft, "Azure Digital Twin Concepts and Models," [Online]. Available: <https://learn.microsoft.com/en-us/azure/digital-twin/concepts-models>. [Accessed: March 25, 2023].
- [55] OpenAI. (2022). "Sample JSON Code for Data Manipulation." [Online]. Available: <https://www.example.com/sample-json-code>. [Accessed: June 8, 2023].
- [56] "Domain Name System (DNS) - Definition," TechTerms. [Online]. Available: <https://techterms.com/definition/dns>. [Accessed: March 10, 2023].
- [57] "Command Line Interface (CLI) - Definition," TechTerms. [Online]. Available: <https://techterms.com/definition/cli>. [Accessed: March 10, 2023].
- [58] Microsoft. "Azure IoT Hub." [Online]. Available: <https://azure.microsoft.com/en-us/services/iot-hub/>. [Accessed: March 10, 2023].
- [59] "Application Programming Interface (API) - Definition," TechTerms. [Online]. Available: <https://techterms.com/definition/api>. [Accessed: March 11, 2023].
- [60] Microsoft. "Azure Functions." [Online]. Available: <https://azure.microsoft.com/en-us/services/functions/>. [Accessed: March 10, 2023].
- [61] R. A. P. Rajan, "Serverless architecture-a revolution in cloud computing," in *Tenth International Conference on Advanced Computing (ICoAC)*, IEEE, pp. 88–93, 2018
- [62] Microsoft. "Event Grid event subscriptions in Azure." [Online]. Available: <https://learn.microsoft.com/en-us/azure/event-grid/create-view-manage-event-subscriptions>. [Accessed: March 11, 2023].
- [63] "Software Development Kit (SDK)," in *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, IEEE Std 610-2019, pp. 667-668, 2019

- [64] B. O. Tayo, "Simplicity vs Complexity in Machine Learning — Finding the Right Balance," 11 Nov 2019. [Online]. Available: <https://towardsdatascience.com/simplicity-vs-complexity-in-machine-learning-finding-the-right-balance-c9000d1726fb>. [Accessed Oct 2021].
- [65] Microsoft. "Azure Monitor log streaming." [Online]. Available: <https://learn.microsoft.com/en-us/azure/azure-functions/streaming-logs>. [Accessed: March 12, 2023].