UNIVERSITY OF TWENTE.

BRIDGING THE SEMANTIC GAP: SEMANTIC INTEROPERABILITY FOR IOT INTEGRATIONS ON IPAAS ARCHITECTURES

Master Thesis

Business Information Technology

Vincent Kat (s2896184)
12 February 2024
Master BIT - University of Twente

Supervisors:
Dr.ir. Erwin Folmer (UT - Faculty of BMS)
Dr.ir Marten van Sinderen (UT - Faculty of EEMCS)
Samet Kaya MSc. (eMagiz - Manager Expert Services)

Version history

Below, a table is given with an overview of the different versions of this document. This to get more clearance about which version every involved actor has.

Version	Distributed to	Distribution date	Comment
v0.1	Samet Kaya (eMagiz Supervisor), Erwin Folmer (UT Supervisor 1), Marten van Sinderen (UT Supervisor 2)	23-10-2023	First version of the research project.
v0.2	Samet Kaya (eMagiz Supervisor), Erwin Folmer (UT Supervisor 1), Marten van Sinderen (UT Supervisor 2)	22-12-2023	Next version of the research for feedback.
v0.3	Erwin Folmer (UT Supervisor 1), Marten van Sinderen (UT Supervisor 2)	05-01-2024	More complete draft version.
v0.4	Erwin Folmer (UT Supervisor 1), Marten van Sinderen (UT Supervisor 2)	14-01-2024	Processed feedback, version for Green Light.
v0.5	Samet Kaya (eMagiz Supervisor)	05-02-2024	Version for final feedback
v1.0	University of Twente, eMagiz	12-02-2024	Final version, uploaded to essay.utwente.nl.

Acknowledgements

I would like to express my deepest appreciation to all those who provided me the possibility to complete this project. A special gratitude I give to the project sponsor, eMagiz, who made this project possible. Their belief in my potential and their willingness to give me a chance has been a significant factor in my journey.

I am deeply indebted to my supervisor Samet Kaya whose help, stimulating suggestions, and encouragement helped me in all the time of research for and development of this project. His dedication, time and efforts have been a tower of strength throughout this period.

I would also like to express my gratitude towards Marten van Sinderen and Erwin Folmer for their academic guidance, insightful comments, and constructive criticisms during the planning and development of this research work. Their willingness to give their time so generously has been very much appreciated.

Finally, I wish to thank all who, directly or indirectly, have made it possible for me to complete this work.

Abstract

This paper explores the rapidly expanding field of the Internet of Things (IoT), focusing on the integration and interpretation of the big amounts of data generated by IoT devices. The study highlights the crucial role of Semantic Interoperability in enhancing the value of IoT data for organizations and identifies Integration Platform as a Service (iPaaS) architectures as a potential solution for facilitating IoT data integrations. However, the challenge is in incorporating Semantic Interoperability within these iPaaS architectures.

The research aims to design an iPaaS architecture which leverages Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations. This will enhance the development efficiency, governance, and seamlessness of IoT integrations. The study contributes to the fields of iPaaS, IoT, and Semantic Interoperability by presenting a treatment enabling iPaaS architectures to semantically interoperate heterogeneous IoT appliances.

The research project followed the Design Science Research Method (DSRM) as mentioned by Peffers et al. (2007). With the help of quantitative evidence, as well as interviews, this study demonstrates how Semantic Interoperability could be incorporated into iPaaS architectures for IoT integrations.

Based on the quantitative evidence, a treatment has been designed, which has been validated with the relevant stakeholders before the start of the development. The developed Proof of Concept consisted of an iPaaS platform, Kafka Broker, Stream Processor, and an IoT Gateway. The Proof of Concept enabled the iPaaS to semantically enrich the IoT integrations based on relevant imported ontologies.

Three cases proved how Semantic Interoperability enhanced the development efficiency, governance, and seamlessness of IoT integrations on an iPaaS architecture. The treatment has been further validated by domain experts as well, which further ensures the validity of the designed treatment.

It can be stated that Semantic Interoperability has the expected positive effects on IoT integrations. With Semantic Interoperability, the Integration Specialist can develop IoT integrations more efficiently, with the data in a clear structured semantic notation, the governance over the IoT integrations is improved, and with the data following the standardized semantics, the seamlessness of the IoT integrations is improved.

Possible future research could focus on enabling Semantic Interoperability for other relevant appliances on iPaaS architectures. Further, a more intelligent matching algorithm could be established to get better results while creating IoT integrations.

Contents

1. Introduction

Currently, the field of Internet of Things (IoT) is rapidly expanding. The amount of connected IoT devices is expected to almost double in the coming seven years (Transforma Insights, 2023). These IoT devices generate enormous amounts of data, which can hold a big value for organizations. This because this data can provide insights, enable automation, and support the decision-making processes. The real potential of IoT data can only be realized when it is effectively integrated and interpreted in a meaningful way.

Semantic Interoperability plays a crucial role in enhancing the meaning and value of IoT data for these organizations (Widell, 2020). Semantic Interoperability refers to the ability of two or more limited systems or components to exchange information with a shared and common meaning in an unambiguous manner.

Integration Platform as a Service (iPaaS) architectures can be a promising solution for facilitating these IoT data integrations, but the incorporation of Semantic Interoperability remains a challenge. Such an iPaaS architecture is described as a cloud-based solution for creating, managing, and governing integration flows.

iPaaS provides a platform that enables the connection and communication between different applications, systems, and data sources. It offers capabilities such as data transformation, routing, and orchestration, which are essential for integrating and managing IoT data. Ensuring that IoT data is not only integrated, but also interpreted correctly across different applications and systems, requires addressing issues related to data semantics, ontologies, and standards.

This research project aims to answer the research question on how to design an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations, to enhance development efficiency and governance in IoT application integrations. This study contributes to the field of iPaaS, IoT, and Semantic Interoperability by presenting a treatment which enables iPaaS architectures to semantically interoperate the heterogenous IoT appliances.

First, the background of the research is discussed with the help of previous conducted initial research. This initial research investigated the state of the art and the challenges in the field of Semantic Interoperability, IoT, and iPaaS.

In the next chapter, the research design is given, involving the structure of the research and the scope of the project. In the fourth chapter, the objectives of the research are stated, including an exploration of existing solutions, stakeholders, and success criteria.

In the fifth chapter, the results are given. These results are the answers to the first three research questions. The first question, on how IoT data integrations can be incorporated in iPaaS architectures, provides insights into IoT and iPaaS architectures. The second question, on how Semantic Interoperability can be incorporated in iPaaS architectures, provides a further foundation for the design of an iPaaS architecture which enables Semantic Interoperability for IoT integrations. Finally, the answer on the third research question, on what the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across different systems are, is given by the design of a treatment.

In the sixth chapter, the designed treatment is demonstrated with the help of a Proof of Concept. The Proof of Concept involves an iPaaS platform enabling Semantic Interoperability for IoT integrations.

In the seventh chapter, the whole treatment validation and evaluation is discussed with the help of the validation and evaluation methods of Wieringa & Morali (2012). Further, the treatment is also validated with the help of three case studies. These cases involve relevant IoT integration cases for the company this research is conducted at. Finally, the conclusion and discussion are stated involving the contributions, limitations, and recommendations of this study.

2. Background

In this chapter, the background of the study is given. First the context of this study is discussed, next, the methodology used for the literature review is given, followed by the results and the conclusions of the literature review. This literature review has been conducted prior to this research project and involved initial research into the state of the art and the challenges of Semantic Interoperability, IoT, and iPaaS.

2.1 Context

For identifying the problem for the literature review, background research has been conducted which provided insights in the challenges of IoT. This background research concluded that valuable data is being generated due to the rapid expansion of the amount of IoT devices (Transforma Insights, 2023). This highlights the need for effective integration and interpretation of this data, which can be done by achieving Semantic Interoperability (Čolaković & Hadzialic, 2018). Semantic Interoperability ensures a common data understanding between two or more systems (Heiler, 1995). A promising solution for integrating IoT data could be iPaaS, but this faces challenges in incorporating Semantic Interoperability (Cestari et al., 2020).

The three goals of the conducted review were: 1) Getting knowledge about the state of the art in the fields of Semantic Interoperability for IoT data integrations on iPaaS, 2) Discovering the key challenges and limitations in achieving Semantic Interoperability for IoT data integrations on iPaaS, and 3) Getting a clear research foundation for the final master thesis project based on the outcomes of the literature review.

The research question that addresses these goals was “What is the state of the art, key challenges, and limitations in achieving Semantic Interoperability in IoT data integrations on iPaaS?”, and consisted of two sub questions: 1) “What is the state of the art in the fields of Semantic Interoperability for IoT Data Integrations on iPaaS?”, and 2) “What are the existing challenges and limitations in achieving Semantic Interoperability for IoT Data Integrations on iPaaS?”.

2.2 Methodology

The literature review that has been conducted prior to the research project followed the Systematic Literature Review (SLR) method of Okoli (2015), which combines best practices for literature reviews from the Information Systems and the Management field.

The protocol of the literature review consisted of four steps. The first step was retrieving resources based on given keywords in the Web of Science, which resulted in 753 resources. After applying inclusion and exclusion criteria 128 resources were left.

The next step was deleting irrelevant and duplicate resources, this resulted in 108 resources. Based on this filtered list without duplicate resources, the titles of the resources were screened based on their relevance, resulting a total of 79 resources.

With the filtered list of resources, backwards and forwards reference searches have been conducted. This resulted in 107 resources, who later have been screened on their contents. This resulted in a total of 32 scientific resources which have been used during the review. The important takeaways of the review are stated in the next chapter.

2.3 Results of Literature Review

2.3.1 State of the Art

State of the Art in IoT

The Internet of Things (IoT) can be described as a global infrastructure that connects physical and virtual objects through information and communication technologies. The IoT paradigm follows three key visions: Things oriented, Internet oriented, and Semantic oriented. In this paradigm, the internet serves as the network to which these various 'things' such as sensors, actuators, and processors, are connected. The importance is also placed on the semantics, highlighting the importance of the meaning of the data.

Furthermore, the IoT is composed of six elements: Identification, Sensing, Communication, Computation, Services, and Semantics. The technology underlying IoT can be divided into three layers: Things, Connectivity, and Cloud.

Various standardization organizations play a crucial role in establishing IoT standards. Notably, long-range communication standards for IoT devices include Low-Power Wide Area Networks (LPWANs) like Sigfox, LoRaWAN, and NB-IoT. Infrastructure protocols for IoT encompass Routing Protocol for Low Power and Lossy Network (RPL), 6LoWPAN, IEEE 802.15.4, Bluetooth Low-Energy, Zigbee, EPCglobal, LTE-A, and Z-Wave.

Protocols used in IoT applications include the Constrained Application Protocol (CoAP), the Message Queue Telemetry Transport (MQTT) protocol, the Extensible Messaging and Presence Protocol (XMPP), the Advanced Message Queuing Protocol (AMQP), and the Data Distribution Service (DDS) protocol.

IoT has a wide range of applications, including logistics, healthcare, environments, homes, offices, cities, industries, firefighting, mining, retail, agriculture, automotive, commerce, and education. Most of these applications fall under the Industrial Internet of Things (IIoT) or Industry 4.0 (I4.0) concept, which uses technologies like IoT to enhance production environments by reducing costs, increasing efficiency, enabling remote operations, and more.

The value added by IoT is its ability to integrate different digital components, leading to the development of new features, services, and business models that enhance efficiency, convenience, and user experience. Moreover, IoT offers new opportunities for companies to explore.

Lastly, the concept of the Web of Things (WoT) involves connecting physical objects to the digital world with three defined integration patterns: Direct Integration, Gateway Integration, and Cloud Integration.

State of the Art in Semantic Interoperability

Interoperability plays a crucial role in enabling systems and components to effectively exchange and utilize information. Within the context of the IoT, Semantic Interoperability becomes important, ensuring that constrained IoT devices can share data with a clear and shared understanding.

The protocol for achieving Semantic Interoperability involves three key steps: describing the exchanged information in a semantic manner, defining the semantics of the interaction, and providing semantic descriptions for the other relevant information. There are many advantages of implementing Semantic Interoperability like enhancing overall interoperability specifications, mitigating interpretation challenges, and facilitating system maintenance and expansion.

To meet the requirements for Semantic Interoperability, a strategic approach is needed. This involves co-creation and separating concerns, establishing necessary knowledge boundaries, adopting

modular design patterns, thoroughly evaluating specifications, and addressing industry deployment considerations.

Furthermore, when aiming to bridge the semantic gap, specific requirements must be met. This involves using standardized semantics, seamlessly integrating systems, ensuring information completeness, optimizing data volume, managing information flow, ensuring individual security and privacy, and leveraging existing IoT interoperability solutions.

To effectively implement Semantic Interoperability in the field of IoT, several semantic web technologies could provide a solution, such as ontology modelling, ontology reuse, and ontology validation tools. The W3C Semantic Sensor Network (SSN) ontology stands out as a widely recognized standard for describing IoT data.

Lastly, numerous ontologies and frameworks can be used to facilitate Semantic Interoperability. These include the IoT Resource Name System (IoT RNS), Cross-Domain Ontology (CDOnto), domOS Common Ontology (dCO), Data and Information Interoperability Model (DIIM), W3C SSN, W3C WoT, oneM2M Base Ontology, MyOntoSens, and SAREF. Each of these ontologies contributes to the broader goal of seamless information exchange in the IoT landscape.

State of the Art in iPaaS

Integration Platform as a Service (iPaaS) is a part of the bigger "X as a Service" movement, which revolutionizes cloud-based software delivery to customers via the internet. iPaaS can be described as a suite of cloud services that enable users to easily create, manage, and monitor integration flows. iPaaS connects applications or data sources without the help of hardware or middleware management.

The use of iPaaS can be categorized into three key areas: ensuring data consistency across applications, automating complex business processes, and creating composite services exposed as APIs or events.

Within the iPaaS landscape, three architectural variants can be seen: Cloud Development & Cloud Execution, Cloud Development & Local Execution, and Local Development & Local Execution. The design of an integration platform should follow the suggested reference architectures to benefit both practitioners and academics in the field of integration platform development and analysis

iPaaS is composed of four main components: Integration processes, Data Mapping, Pre-built Connectors, and the necessary support for these components.

The roots from iPaaS lies in the tradition of Enterprise Application Integration (EAI), iPaaS serves as the cloud-based evolution of EAI, offering reduced complexity, user-friendliness, quicker integration of new applications, and lower maintenance costs.

Various factors influence iPaaS adoption, including the standardization of data models, connector usability and diversity, and the commitment of the platform to data protection, security, and transparency.

The iPaaS business model can be seen as a multi-sided platform, creating value for different user groups.

Key requirements for an effective iPaaS solution include a wide array of connectors, customizable connectors, intelligent data mapping, comprehensive documentation, testing options, a robust data hub, and robust reporting capabilities.

Furthermore, in the field of IoT integrations, Edge Computing could offer a promising solution. This approach positions computing resources closer to the devices instead of centralizing them in massive datacenters. Edge Computing facilitates more efficient IoT integrations.

2.3.2 Challenges and Limitations

There are a few challenges in IoT. Security challenges include the confidentiality of the data, and tamper-resistance hardware. Interoperability challenges result from the many different manufacturers of IoT devices implementing their own interfaces and protocols.

Challenges that could occur while incorporating Semantic Interoperability in IoT data integrations include the spread of project-specific ontologies and difficulties in normalizing concepts between different ontologies. Challenges that have been identified in iPaaS include data security, data performance, lack of standardization, and reliability.

2.4 Motivation for the research project

The motivation for the research project originates from the results of the literature review. The research aims to create seamless integrations between different IoT data sources in iPaaS. It is worth noting that there is currently a lack of research in the field of iPaaS that incorporates Semantic Interoperability for IoT data integrations. Despite the rapid growth of IoT technology, there remains a gap in the understanding on how to effectively integrate IoT data in iPaaS platforms while ensuring Semantic Interoperability. Moreover, there continues to be challenges and limitations in the field of Semantic Interoperability for IoT Data Integrations in iPaaS. These challenges are related to data standardization, ontology mapping, and ensuring the seamless integration of information between different IoT devices and platforms.

iPaaS could be the key enabler for seamless integration between the IoT devices. This will make sure the data can flow between different IoT systems and devices without any problems. On the other hand, Semantic Interoperability can be seen as the missing link between IoT and iPaaS. Semantic Interoperability will ensure that, when data moves between two systems, it will be in the same context and meaning.

Added value

The added value of this research project is to address the issue in the IoT domain. This issue can be seen as a limitation to the seamless integration of data from various IoT sources. This is an essential aspect of IoT, as it ensures that the data generated by different devices can be effectively used, analysed, and leveraged for various applications.

The first aspect of the added value is the enhancing of IoT integrations. By focusing on creating a more seamless connection between different IoT sources, the research project aims to enhance the ability of organizations to integrate and use data from all the different IoT devices. This will result in a better view of their IoT landscape.

The second aspect of the added value is the improvement of efficiency. The introduction of iPaaS as a key enabler for IoT data integration can improve the efficiency of the development of IoT integrations. This means that organizations can work more effectively with IoT data without problems, which can lead to cost savings and improved operations.

The third aspect of the added value is the semantics. The incorporation of Semantic Interoperability ensures that data transferred between different systems remains a consistent meaning and context. This is important for the analysis and decision-making process. This because it prevents misunderstandings and misinterpretations of the data.

A fourth aspect of the added value is the improvements of current iPaaS architectures. Incorporating Semantic Interoperability in iPaaS architecture will contribute to the iPaaS technology, making it more adaptable to IoT integration challenges.

The last aspect of the added value is the governance of development. The objectives of the research project include enhancing governance in IoT application integrations. This means that organizations will have better control and oversight of their IoT data integration processes. This governance is important for compliance, security, and overall IT management.

3. Research design

3.1 Research Questions

Based on the conducted literature review, a research design can be stated. The question for this research addresses the insights that have been gathered from the literature review. This research question is stated below.

How to design an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations, to enhance development efficiency and governance in IoT application integrations?

This main research question is composed of four sub-questions which are stated below:

***(SQ1)* How can IoT data integrations be incorporated in iPaaS architectures?**
The objective of the first question is to investigate how IoT data integrations can be incorporated into iPaaS architectures. This objective focuses on understanding the technical aspects of integrating IoT sources into the iPaaS architecture.

***(SQ2)* How can Semantic Interoperability be incorporated in iPaaS architectures?**
The objective of the second question is to investigate how Semantic Interoperability can be incorporated into iPaaS architectures. This objective focuses on the concepts and technologies required to ensure Semantic Interoperability when integrating data across iPaaS architectures.

***(SQ3)* What are the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across different systems?**
The objective of the third question is to identify the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations. This objective aims to define the specific attributes and features necessary for such an architecture. This question will involve a solution design for an iPaaS architecture which will support Semantic Interoperability for IoT data integrations.

***(SQ4)* What can be the impact of iPaaS architectures that leverages Semantic Interoperability to seamlessly integrate IoT data across systems based on three specific cases?**
The objective of the final question is to demonstrate the artifact based on three specific cases, and to assess the potential impact of iPaaS architectures that leverage Semantic Interoperability in the context of these cases. This objective seeks to understand the practical benefits and consequences of implementing the proposed architecture.

To get a better overview of this research, a conceptual framework has been illustrated in figure 1. This framework is built upon the interaction of several key concepts, including Semantic Interoperability, IoT Data Integrations, and iPaaS Architectures.

IoT Data Integrations share a positive relationship with iPaaS Architectures. This relationship is partially mediated by Semantic Interoperability. This means that the quality and efficiency of IoT Data Integrations in iPaaS architectures are partially explained through Semantic Interoperability.

Semantic Interoperability plays a partial mediation role. This variable partially explains the positive relationship between IoT Data Integrations and iPaaS Architectures. This means that when Semantic Interoperability is effectively implemented, it enhances the integration of IoT data in iPaaS architectures.

iPaaS Architectures play a central role within the framework. First, they contribute positively to Efficiency within IoT Data Integrations of Clients. This suggests that by employing iPaaS Architectures, organizations can manage their IoT data processes more efficiently. Next, iPaaS Architectures contribute positively to Governance within IoT Data Integrations of Clients. This suggests that by employing iPaaS Architectures, organizations can streamline their IoT data processes more efficiently. Finally, iPaaS Architectures are associated with the Seamless Integration of Clients' IoT Data. This indicates that these architectures facilitate a smoother and more effective integration of IoT data for clients, which in turn contributes to the overall success of IoT implementations.

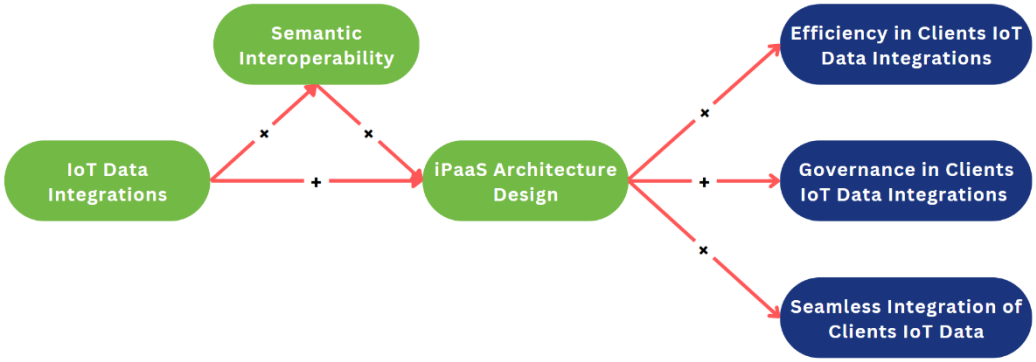

Figure 1: Conceptual Framework

3.2 Research Methodology

There are different research methodologies available for a design science project. A selection of three design science methods is given to select the best fitting method for this research project. According to Sonnenberg & Vom Brocke (2012), design science theories consist of two high level activities: Building and Evaluating.

The first selected method is the Action Design Research (ADR) by Sein et al. (2011). ADR is a research method used to create practical design knowledge by building and assessing IT tools in a real-world organization. It tackles two major challenges: 1) solving a specific issue within an organization by intervening and evaluating, and 2) establishing and evaluating an IT tool that can handle similar issues. This approach emphasizes constructing, intervening, and evaluating an IT tool that is not only compatible with the researchers' theories, but also focuses on user input and real-world usage. Looking at the statement of Sonnenberg & Vom Brocke (2012), a clear division of the Build and Evaluate activities can be seen. The Build activity consists of the Problem Formulation and the Building of the artifact. Next, the Evaluation consists of Intervention and Evaluation, Reflection and Learning, and Formalization of Learning. An overview of the Action Design Research Method by Sein et al. can be seen in figure 2.

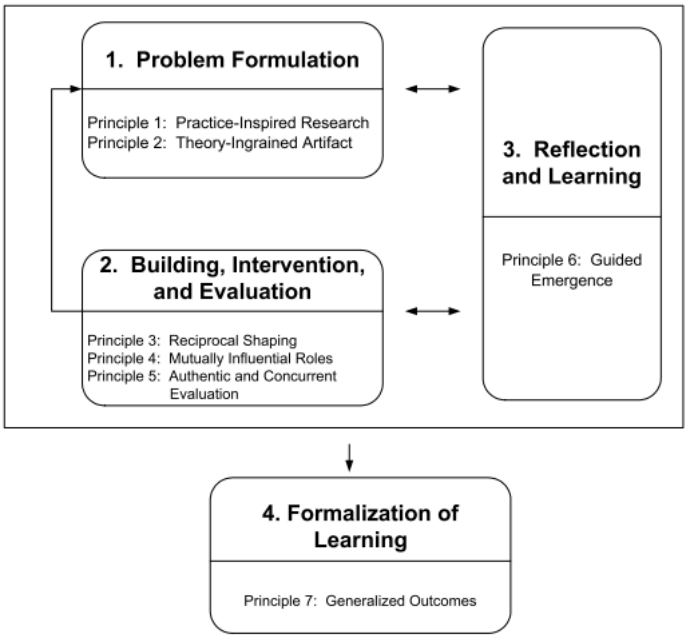

Figure 2: Overview of the Action Design Research Method by Sein et al. (2011)

The second method that could be applied in this project is the Design Science Methodology of Wieringa (2014). This method is divided into three main parts: problem investigation, treatment design, and treatment validation, which together make up the design cycle. This cycle is repeated multiple times during the project. The design cycle is part of a bigger cycle known as the engineering cycle. In this larger cycle, the results of the design cycle, which is a validated treatment, are put into action in the real world and then evaluated.

Wieringa also adds that the evaluation of the implementation may lead to the problem investigation of a new engineering cycle. Additionally, the management of research and development processes, including decision-making, stakeholder alignment, and resource allocation, are separate aspects and are not within the scope of the engineering and design cycle. Looking back at the Build and Evaluate statement of Sonnenberg & Vom Brocke (2012), a clear division can be stated for this theory as well. The Build activity involves the Problem investigation and the Treatment Design, while the Evaluation activity consists of the Treatment Validation and the Treatment evaluation. An overview of the Design Science Cycle by Wieringa can be seen in figure 3.

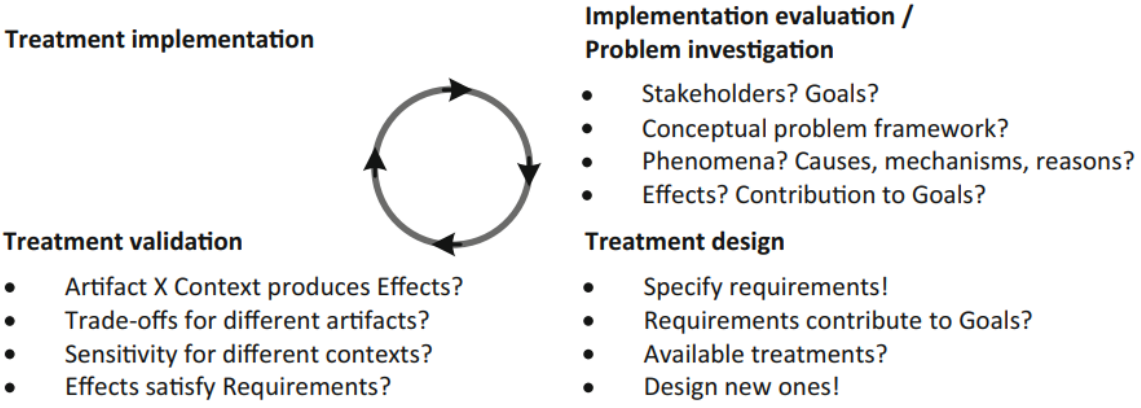

Figure 3: Design Science Cycle by Wieringa (2014)

The last selected method that can be applied is the Design Science Research Methodology (DSRM) by Peffers et al. (2007). The DSRM provides a process for conducting a design science research project, which can be seen in figure 4. The process consists of different steps which include 1) the problem identification and motivation, 2) defining the objectives of a solution, 3) designing and development of the solution, 4) demonstration of the solution, 5) evaluation of the solution, and 6) communication of the solution. Based on the outcomes of step 5 or 6, it can be chosen to iterate back to step 2 or 3. Referring to the Build and Evaluate activities of Sonnenberg & Vom Brocke (2012), a clear division can be seen as well. The Build activities consist of identifying the problem, defining the objectives, and designing and developing. While the Evaluate activities consist of demonstrating, evaluating, and communicating.

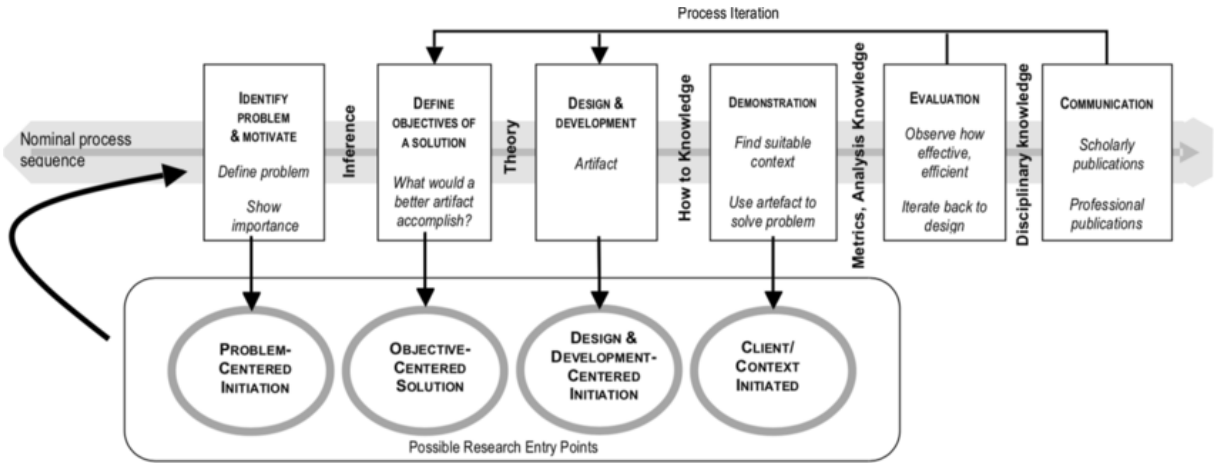

Figure 4: Design Science Process by Peffers et al. (2007)

To select the best method, a comparison of the given methods is made. The ADR approach a is a problem-driven approach, which aims to build design principles more based on iterative client cycles for multiple different clients (Wieringa & Morali, 2012). This research will not actually be in the context of multiple different clients; thus, this method is not a perfect fit.

The method of Wieringa provides an overview of a cycle with the focus on the design aspect. Although design is an important factor in this project, proving that the design works with a Proof of Concept also has the importance. That is why the selected method for this research project will be the Design Science Research Method (DSRM) as mentioned by Peffers et al. (2007). This method provides a commonly accepted framework for carrying out design research with a process approach while the other two stated methods provide a cycle approach (Vom Brocke & Siedel, 2012). Because this project has a fixed duration, and because the executive researcher prefers to use a more structured approach, a process-based method has been chosen. This approach provides more structure than cycle centred approaches, while still providing the option for an iterative approach within a guiding process.

3.3 Structure of the research

The stages of the Design Science Research process, with their relation to the research questions, are stated below. An illustrated overview of the planning can be seen in figure 5.

Stage 1: Problem Identification & Motivation

In the first stage, the context and motivation of this research will be presented. This will be demonstrated by the literature review conducted in the research topics project. The importance of IoT data integrations and Semantic Interoperability in iPaaS architectures will be explained, and existing challenges or gaps in the literature relating to these topics will be discussed.

Stage 2: Objective Definition
During the second phase, existing solutions will be explored. Further, the research objectives and questions will be formalized, providing an overview of the aims of this study. The success criteria and the requirements for the artifact will also be defined.

Stage 3: Design & Development
This stage involves the design and development of the proposed artifact. To address SQ1 and SQ2, a supporting literature study will be conducted to answer these questions. Based on the answers, a general applicable iPaaS architecture design will be presented in SQ3. Further, a Proof of Concept will be developed to create a platform to answer the case study part, SQ4, of the research. This Proof of Concept will be tested, followed by refinements based on the test outcomes.

Stage 4: Demonstration
In this stage, it will be demonstrated how the solution addresses the given problem. This involves presenting the Proof of Concept, showing the architecture, explaining its functionality, and describing its capabilities.

Stage 5: Evaluation
This stage addresses the evaluation to answer SQ4 based on the impact of the developed artifact. This evaluation will measure the effectiveness and advantages of the proposed solution, based on the outcomes of the case study conducted in SQ4. An iteration over the objectives, requirements, design, or development may be executed to improve the results of the project.

Stage 6: Communication
The last stage involves the distribution of the findings and results. The master thesis will be written, offering detailed responses to each research question based on the work conducted in the described stages. Next, the implications of the findings and their contributions to the field will be discussed. A possible iteration over the objectives, requirements, design, or development may be conducted to further improve the results of the project.

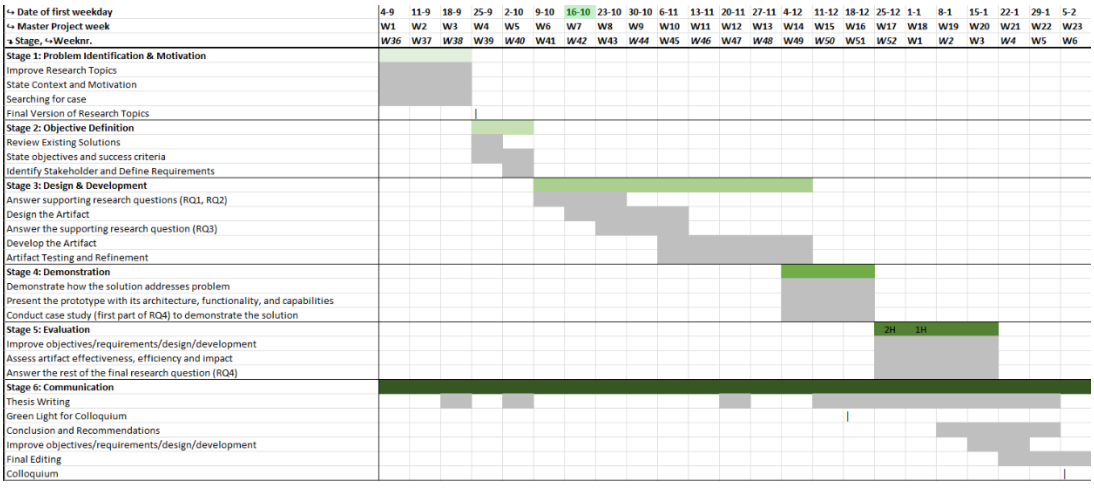

Figure 5: Illustrated Overview of the Project Planning

3.4 Scope of the project

The project scope involves an investigation into the design and development of an iPaaS architecture using Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations. Following the planning, the duration of the research project aims to take around 23 weeks, which means that the project aims to be finished in the first week of February 2024.

The research aims to be generally applicable for iPaaS architectures. This is why the design will be able to support general iPaaS architectures. For proving the concept of this project, the Proof of Concept will be based on the requirements of the stakeholders at eMagiz. This is the company at which this study is conducted.

The first research question of this study, SQ1, will investigate how specific methods for IoT data integrations in iPaaS platforms could be incorporated. This will investigate at least the main three methods for integrating IoT on an iPaaS architectures. These methods have already been identified in the literature review, meaning that this study will investigate how to enable an iPaaS to use these methods. As mentioned before, this needs to be generally applicable for iPaaS architectures.

The second research question of this study, SQ2, will investigate how specific methods for Semantic Interoperability in iPaaS platforms could be incorporated, with the focus on IoT data integrations. This question will investigate at least the main three methods for implementing Semantic Interoperability for IoT on an iPaaS architectures. Some methods have already been identified in the literature review, meaning that this study will look on how to enable an iPaaS to use these methods. As mentioned before, this needs to be generally applicable for iPaaS architectures.

Based on the results of these two research questions, an iPaaS architecture will be designed. This designed architecture will enable IoT integrations on iPaaS architectures with the help of Semantic Interoperability. The design will involve a target architecture, functional components, processes, and user interfaces to answer the third research question, SQ3, about the characteristic of these architectures. This design aims to be generally applicable to iPaaS architectures.

A Proof of Concept will be developed, following the created solution design. This Proof of Concept will involve an extension of existing iPaaS architectures, which enables Semantic Interoperability for IoT data integrations. Because this research project is conducted at the company which provides the eMagiz platform, the requirements will be based on the stakeholders within the eMagiz organization. The concept will be demonstrated through three relevant cases, all aligning with the customer segments of eMagiz. The Proof of Concept will be developed using the low-code development platform Mendix, which is used within the eMagiz organization.

The three cases that will be treated in this study need to validate that the treatment tackles the given problem. These cases will be applied using physical or virtual (simulated) IoT devices, which will be integrated with enterprise applications with the help of iPaaS. The treatment needs to enable the Semantic Interoperability for these IoT integrations. It will not directly connect to the devices but to the IoT platforms / gateways using the MQTT protocol. It is chosen to use MQTT because as of today, MQTT seems to be leading the way in IoT applications (Priyadarshi & Behura, 2018). Further, Yudidharma et al. (2023) state that MQTT is a commonly used messaging protocol which performs better than other protocols as it has wide network bandwidth and a low size of transferred packets.

Finally, the results of the study will be evaluated based on the effectiveness, efficiency, impact, and validity to answer the fourth research question, SQ4. The results will be processed in the thesis report.

4. Objectives

In this chapter, the objectives for the research project are given. This is the second stage as mentioned in DSRM by Peffers et al. (2007). First an analysis of the existing solutions will be given, followed by an analysis of the stakeholders, objectives, success criteria, and requirements.

4.1 Existing solutions

4.1.1 Identifying existing solutions

To identify the existing solutions for the given problem, the Magic Quadrant of Gartner will be used. The Magic Quadrant positions technology players within a specific market on a matrix, or quadrant, based on their completeness of vision of the technology, and their ability to execute this technology (Gartner, sd). This matrix consists of four different plots, explained below. An overview of the Gartner Magic Quadrant can be seen in figure 6.

Niche Players
These companies are like specialists in a specific niche or small segment of the market. They have a narrow focus and might not have a complete vision of the entire market. Their ability to put their technological plans into action is somewhat limited. They can be referred to as experts in a small corner of the market that struggle to expand beyond that.

Visionaries
The Visionaries are the dreamers of the market. They understand where the market is heading, or they might even have innovative visions for changing the market. However, they may not be the best at executing their visionary ideas when it comes to technology. They can be referred to as the creative thinkers who sometimes struggle to turn their ideas into reality.

Challengers
The challengers are the doers in the market. They are strong when it comes to executing their technologies compared to other players. However, they may lack a clear vision of how the market will evolve. They can be referred to as good runners in a race, but they might not know exactly where the finish line is.

Market Leaders
These are the top companies in the market. Market Leaders not only have a clear and complete vision of where the market is heading, but they can also effectively execute their technological plans. They can be referred to as the all-round champions who know where they are going and how to get there.

Figure 6: Gartner Magic Quadrant by Gartner (sd)

4.1.2 Identified existing solutions

To get a clear understanding of the current solutions in the iPaaS field, a global overview of iPaaS solutions is given in figure 7. This overview is provided by Gartner, which published the Gartner Magic Quadrant for integration Platform as a Service (Gartner, 2023).

Figure 7: Gartner Magic Quadrant iPaaS by Gartner (2023)

Niche players

The niche players in the field of iPaaS are described below. These companies are the specialists in a specific niche of the market. They might not have a complete vision of the entire market, with a limited ability to put their technological plans into action.

Celigo (Integrator.io)

Celigo offers integrator.io, which combines iPaaS, Business Process Management (BPM), and process mining for integration and automation. There are prepackaged integrations and connectors on the platform, with customizable integration templates. The operations of Celigo are mostly in the United States, with clients in the retail, technology, manufacturing, and services sectors. Gartner named the customer experience, marketing strategy, and the sales execution/pricing as the strengths of the platform, while the geographical presence, product/services, and market understanding are the cautions of the platform (Gartner, 2023).

Huawei (ROMA Connect)

Huawei offers ROMA Connect, which provides integrations for applications, data, Electronic Data Interchanges (EDIs), and events as well as features for API management. The operations of Huawei are mainly focussed in the Asia-Pacific (APAC) region, with a few operations in the Europe, Middle East & Africa (EMEA) region. The clients are in the government, education, manufacturing, smart cities, and energy sectors. Gartner named the viability, customer experience, and innovation as the strengths of the platform, while the geographic strategy, product/services, and market responsiveness are the cautions of the platform (Gartner, 2023).

Talend (Data Fabric)

Talend offers Talend Data Fabric, which provides integrations for data, applications, and APIs. The platform also provides services for data integrity and governance. The operations of Talend are geographically diverse, with clients in the financial services, manufacturing, retail, and insurance sectors. Gartner named the innovation, sales strategy, and vertical/industry strategy as the strengths of the platform, while the sales execution & pricing, customer experience, and offering product strategy are the cautions of the platform (Gartner, 2023).

Frends

The Frends platform provides integration and automation for Integration Specialist and Business Process Analyst. The operations of Frends are mainly focussed in the Europe, Middle East & Africa (EMEA) region, with a few operations in the United States and in the Asia-Pacific (APAC) region. The clients are in the government, education, manufacturing, smart cities, and energy sectors. Gartner named the market responsiveness, vertical/industry strategy, and customer experience as the strengths of the platform, while the geographic strategy, offering product strategy, and innovation are the cautions of the platform (Gartner, 2023).

Visionaries

The visionaries are the dreamers in the market and are described below. They understand where the market is heading but may not be the best at executing their visionary ideas when it comes to technology.

IBM

IBM offers different solutions like IBM App Connect, IBM Cloud, IBM API Connect, and IBM Cloud Pak. The operations of IBM are geographically diverse, with clients in all sectors. Gartner named the innovation, operations, and geographic strategy as the strengths of the platform, while the offering product strategy, marketing execution, and customer experience are the cautions of the platform (Gartner, 2023).

Software AG (webMethods.io)
Software AG offers the webMethods.io, which provides Business to Business (B2B), managed file transfer (MFT), and API integrations. Software AG also offers the Cumulocity IoT platform. The operations of Software AG are geographically diverse, with clients in all sectors. Gartner named the sales execution, product/services, and geographic strategy as the strengths of the platform, while the customer experience, marketing execution, and innovation are the cautions of the platform (Gartner, 2023).

Jitterbit (Harmony)
Jitterbit offers Jitterbit Harmony, which provides the integration for applications, data, Business to Business (B2B), and APIs. The operations of Jitterbit are geographically diverse, with clients in the manufacturing, technology, healthcare, retail, media, education, and nonprofit sectors. Gartner named the offering product strategy, customer experience, and operations as the strengths of the platform, while the sales execution & pricing, marketing strategy, and vertical/industry strategy are the cautions of the platform (Gartner, 2023).

SnapLogic (Intelligent Integration Platform)
SnapLogic offers the Intelligent Integration Platform (IIP), which provides separate solutions for entry-level projects and enterprises. The add-ons of the platform consist of premium Snaps (pre-built, intelligent connectors), API management, and Business to Business (B2B) integrations. The operations of SnapLogic are mainly in the United States, with clients in the technology, retail, manufacturing, healthcare, financial services, and transportation sectors. Gartner named the innovation, vertical/industry strategy, and geographic strategy as the strengths of the platform, while the market responsiveness, market understanding, and customer experience are the cautions of the platform (Gartner, 2023).

Challengers
The challenger is the doer in the market and is described below. They are strong when it comes to executing their technologies compared to other players, but they may lack a clear vision of how the market will evolve.

Tray.io
The Tray.io platform provides the Tray Platform for organizations, and Tray Embedded for the independent solution providers. The operations of Tray.io are mainly focussed on Northern America, with a few operations in the Europe and the Asia-Pacific (APAC) region. The clients are in all sectors, with a focus on the technology sector. Gartner named the sales strategy, customer experience, and market responsiveness as the strengths of the platform, while the offering product strategy, product/services, and market understanding are the cautions of the platform (Gartner, 2023).

Market Leaders:
The market leaders are the top companies in the market and are described below. They not only have a clear and complete vision of where the market is heading but can also effectively execute their technological plans.

Oracle
Oracle offers different solutions like Oracle Integration, Oracle Cloud Infrastructure (OCI), and Oracle IoT Cloud Service. The operations of Oracle are geographically diverse, with clients in all sectors. Gartner named the sales strategy, geographic strategy, and market responsiveness as the strengths of the platform, while the customer experience, offering product strategy, and sales execution & pricing are the cautions of the platform (Gartner, 2023).

Workato (Workspace)
Workato offers Workato Workspace, which provides Line Of Business (LOB)-driven development, and enterprise-level automation and integration. The operations of Workato are geographically diverse, with clients in all sectors. Gartner named the marketing execution, customer experience, and market responsiveness as the strengths of the platform, while the pricing, geographic strategy, and vertical/industry strategy are the cautions of the platform (Gartner, 2023).

SAP (Integration Suite)
SAP offers the SAP Integration Suite, which is part of the SAP Business Technology Platform. The SAP Integration Suite provides integrations for applications, data, processes, and businesses. The operations of SAP are geographically diverse, with clients in all sectors. Gartner named the vertical/industry strategy, sales strategy, and geographic strategy as the strengths of the platform, while the marketing execution, customer experience, and market responsiveness are the cautions of the platform (Gartner, 2023).

Salesforce (MuleSoft Anypoint Platform)
Salesforce offers the MuleSoft Anypoint Platform, which combines API management, integration, and automation. The operations of Salesforce are geographically diverse, with clients in all sectors. Gartner named the vertical/industry strategy, operations, and geographic strategy as the strengths of the platform, while the market understanding, offering product strategy, and sales execution are the cautions of the platform (Gartner, 2023).

Microsoft (Azure)
Microsoft offers the Microsoft Azure Integration Services Platform, which consists of different solutions like Azure logic Apps, Azure API Management, and Azure Service Bus. The operations of Microsoft are geographically diverse, with clients in all sectors. Gartner named the operations, innovation, and market responsiveness as the strengths of the platform, while the vertical/industry strategy, offering product strategy, and sales strategy are the cautions of the platform (Gartner, 2023).

Boomi (AtomSphere)
Boomi offers the Boomi AtomSphere Platform, which consists of different solutions like Boomi Integration, Boomi B2B/EDI Management, and Boomi API Management. The operations of Boomi are mainly focussed on Northern America, with a few operations in the Europe, Middle East & Africa (EMEA) region and the Asia-Pacific (APAC) region. The clients are in all sectors. Gartner named the market responsiveness/record, sales execution & pricing, and vertical/industry strategy as the strengths of the platform, while the offering product strategy, innovation, and business model are the cautions of the platform (Gartner, 2023).

TIBCO Software (Cloud Integration)
TIBCO offers TIBCO Cloud Integration, which provides integrations for applications, data, APIs, Business to Business (B2B), and IoT. The operations of TIBCO are geographically diverse, with clients in all sectors. Gartner named the marketing execution, community & support, and product/services as the strengths of the platform, while customer experience, operations, and offering product strategy are the cautions of the platform (Gartner, 2023).

Based on Gartner's Magic Quadrant, a clear overview of the existing iPaaS solutions is given. The integration platform used in this case study is not mentioned in the Magic Quadrant. eMagiz could be described as a niche player on the iPaaS market. eMagiz has customers in sectors like Construction & Industry, Energy, Food, Financial Services, Transport & Logistics, and Retail. The operations of eMagiz are mainly in the Netherlands.

4.2 Stakeholders

4.2.1 Stakeholder Analysis

To select a stakeholder analysis method, three different methods have been considered. Davis (2023) identified three different stakeholder analysis methods. It was chosen to use the Power-Interest Stakeholder Analysis Matrix by Mendelow (1991). This method uses a matrix with two variables: 1) Level of Interest, and 2) Power. The stakeholders are plotted on this matrix based on these variables. There are four different plots in this matrix, shown in figure 8.

The first plot involves a low level of interest and a low power. The stakeholders in this plot require minimal effort during the stakeholder management process. These stakeholders do not have significant impact on the project.

The next plot involves a high level of interest and a low power. During the stakeholder management process, these stakeholders need to be kept informed. These stakeholders do not have significant impact on the project, but do have

The following plot involves a low level of interest and a high level of power. These stakeholders need to be kept satisfied during the stakeholder management process. These stakeholders can make or break the project.

The last plot involves a high level of interest and a high level of power. These stakeholders are the key players of the project. These stakeholders need to be managed closely to make the project a success.

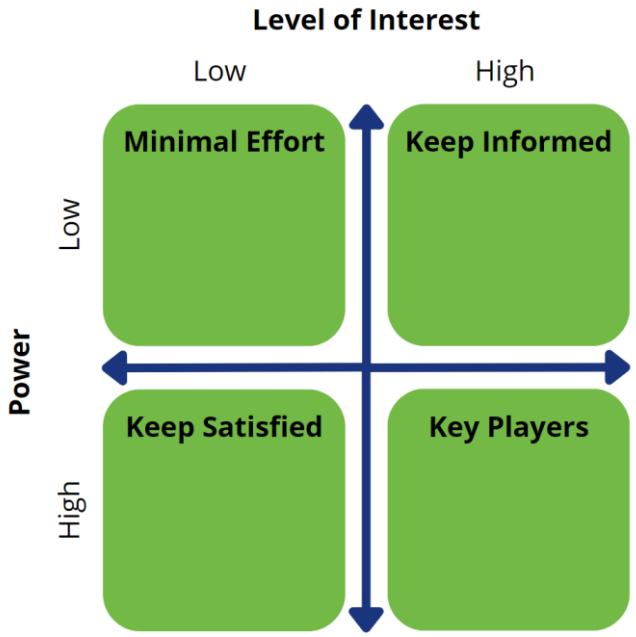

Figure 8: Power-Interest Stakeholder Analysis Matrix

The Stakeholder Matrix by Mendelow provides a clear and simple overview of the stakeholders within a project, with actions on how to manage these stakeholders. It is chosen to use this method because this provides a simple and understandable overview of the involved stakeholders within a project, followed by directions how to involve these stakeholders.

4.2.2 Results

The selection of the methodology used for analysing the stakeholders is described in the chapter above. It was chosen to use the Stakeholder Matrix by Mendelow (1991), which provides two

variables, Interest and Power, to plot stakeholders on a matrix with four plots. The detailed matrix can be seen in figure 9.

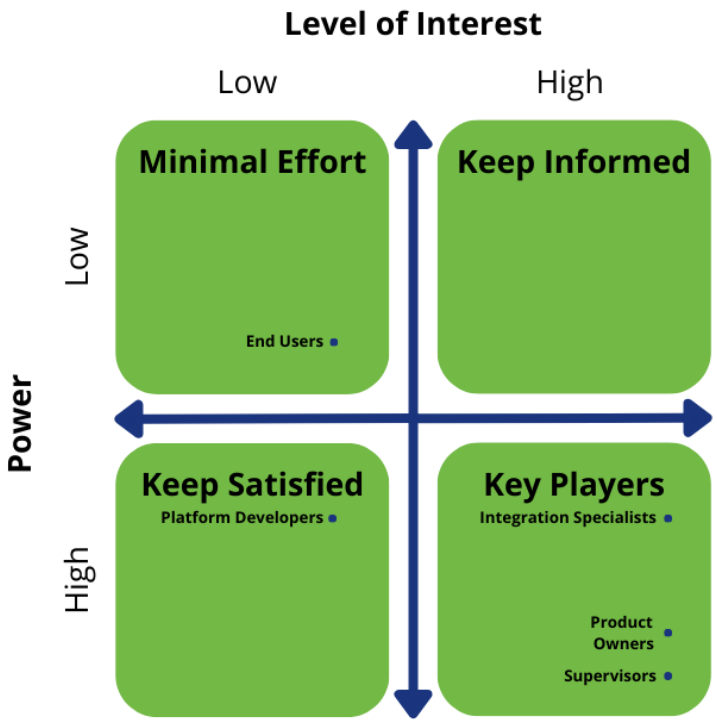

Figure 9: Filled in Stakeholder Matrix for this Research Project

Gartner (2023) mentioned in their Magic Quadrant four different user types that are involved in an iPaaS. These user types are Business Technologist, Integrations Specialist, Software Engineer, and Business Process Expert. These user types can be divided in two stakeholder groups.

Business/End Users

The first stakeholder group which is based on the Gartner Magic Quadrant is the Business User. These stakeholders involve the Business Technologist, Software Engineers, and Business Process Experts user types as mentioned by Gartner. These stakeholders work at the consumer of the iPaaS platform, but they do not directly work within the iPaaS. The platform does provide relevance to their work.

The term Business Technologist could be explained as “an employee who reports outside of IT departments (centralized or business unit IT) and creates technology or analytics capabilities for internal or external business use.” (Gartner, sd). The specific roles within the Business Technology user type could be Software as a Service (SaaS) administrators, Citizen Developers, or Architects.

The term Software Engineer could be explained as “an employee applying engineering principles and knowledge of programming languages to build software solutions for end users.” (Michigan Technological University, 2023). These Engineers do not work directly in the iPaaS because the integrations are realised with the help of the Integration Specialist.

The term Business Process Expert could be explained as “a professional who specializes in analysing and improving business processes to enhance efficiency, productivity, and overall performance.” (Hatmi, 2023). These experts govern the processes, including the integration process.

This stakeholder group has a Low to Moderate interest in the project because this will provide more clearance and governance in combination with improved effectiveness and efficiency. The power of

these stakeholders can be rated as Low to Moderate as well because they stay in contact with the Integration Specialist for implementing the different integrations.

These stakeholders can be classified as the Minimal Effort category on the stakeholder matrix of Mendelow (1991). They do not need to be actively informed, and do not need to be directly kept satisfied, because they do not work in the iPaaS themselves.

Integration Specialists
The second stakeholder group based on the Gartner Magic Quadrant is the Integration Specialist. This stakeholder can be described as "an individual who works to connect different computer systems and applications to ensure seamless communication and data exchange." (Zippia, 2023). The integration specialist directly works in the iPaaS platform and can be either internal or external. Internal refers to the integration specialist working at the platform supplier, external refers to the integration specialist working at the platform consumer.

The interest of this stakeholder group can be seen as High. This because this stakeholder will integrate different IoT appliances with the iPaaS platform. When the integration process can be done more efficiently and effectively, this stakeholder will be more satisfied. The power of this stakeholder can be rated as Moderate to High. This rating is because this stakeholder needs to integrate the different IoT solutions within the proposed architecture, and thus get to make or break the solution.

These stakeholders can be classified as Key Players on the stakeholder matrix of Mendelow (1991). They need to be closely managed to get satisfying results. These stakeholders will be used to validate the proposed solution.

Platform Developers
The Platform Developers are the internal stakeholders working on developing and improving the iPaaS platform with either high-code or low-code. These stakeholders have a Low to Moderate interest in the project because they do not directly have something to do with the project, but if the proposed solution is finally implemented, these stakeholders need to carry out the implementation. The power of these stakeholders can be rated as Moderate to High because they are the ones who need to incorporate the suggested solution into the existing iPaaS architecture.

These stakeholders can be classified as the Keep Satisfied category on the stakeholder matrix of Mendelow (1991). They need to be kept satisfied with the proposed solution to make the outcomes of the project more valuable. These stakeholders will be used to validate the proposed solution.

Product Owner / Manager
The third internal stakeholder group involves the product owners and product managers. These roles help improve the iPaaS product with the help of new features based on the voice of the customer. According to Ankulov (2020), the product manager discovers the needs of the users, prioritizes what needs to be build, and gathers the team around a product roadmap. The product owner is responsible for getting the most value for the product by creating and managing the product backlog, creating user stories, and communicating the voice of the customer.

The interest of this stakeholder can be rated as High. This rating is based on the contribution of the research to the product. The power of this stakeholder can be rated as High. This rating is because these stakeholders set the strategy and vision for the product, which has impact on the eventual execution of the project.

These stakeholders can be classified as Key Players on the stakeholder matrix of Mendelow (1991). They need to be closely managed to get satisfying results. These stakeholders will be used to validate the proposed solution.

Supervisors
The last stakeholder group consists of the supervisors of the project. This group can be divided in internal and external supervisors. The internal supervisor works for the company supplying the project. This supervisor guides the project and provides support with its expertise and experience in the form of feedback and weekly appointments.

The external supervisors are the supervisors on behalf of the University of Twente. These supervisors ensure that the academic part of the project is following the requirements of the University of Twente in the form of feedback and biweekly appointments.

The interest of these stakeholders can be rated as High because these stakeholders have interest in the research itself and in the added value of the outcomes, both academical and practical. The power of these stakeholders can be rated as High because they provide guidance and advice on the research questions, design, and methodology. The external supervisors provide the grade for the project, meaning they must be satisfied to get a successful research project.

These stakeholders can be classified as Key Players on the stakeholder matrix of Mendelow (1991). They need to be closely managed to get satisfying results.

4.3 Objectives

4.3.1 Stating objectives

For stating the objectives of the project, the SMART method by Doran (1981) is used. This method uses five requirements for stating meaningful objectives. Doran stated that the A in the acronym stands for Assignable, and the R for Realistic. There are many interpretations for the individual aspects of the acronym, but a widely adopted adaptation of the SMART methods uses the requirements Achievable and Relevant instead of Assignable and Realistic (Rubin, 2002).

Specific
When setting an objective, it should be clear and well-defined. It should answer the questions of who, what, where, when, and why. This specificity helps in focusing efforts and understanding the desired outcome.

Measurable
Objectives should include quantifiable criteria for success. This means it must be clear how the progress will be measured and when the objectives have been achieved. Measurable objectives provide a clear indication of whether the project is on track or in need of adjustment of the efforts.

Achievable
Objectives should be realistic and attainable within the given resources and constraints. It is essential to consider whether the objective is within the capabilities of the project, and if the necessary tools and resources are available to accomplish the objective.

Relevant
The objective should align with the broader objectives and be relevant to the mission or purpose. It should contribute to the overall strategy or vision and not be a distraction from more important priorities.

Time-bound
Objectives need to have a specific time frame or deadline. This establishes a sense of urgency and prevents objectives from remaining indefinitely. Having a time-bound objective helps with planning and accountability.

4.3.2 Stated objectives

For achieving successful results of this project, six objectives have been stated. The methodology for describing the objectives is the SMART method by Doran (1981), which states that objectives should be Specific, Measurable, Achievable, Relevant, and Time-bound. A more detailed description of the SMART methodology is given in the chapter above.

Objective 1
Analyse and document at least the top three methods for integrating IoT data in iPaaS architectures within two weeks, before the ninth week of the project, by accessing existing literature and collaboration with experts, to gain the necessary knowledge for designing iPaaS architectures with Semantic Interoperability for IoT data integrations.

Objective 2
Research and document at least the top three methods for achieving Semantic Interoperability in iPaaS architectures for IoT data integrations within two weeks, before the ninth week of the project, by accessing existing literature and collaboration with experts, to gain the necessary knowledge for designing iPaaS architectures with Semantic Interoperability for IoT data integrations.

Objective 3
Develop a conceptual iPaaS architecture design with Semantic Interoperability for IoT data integrations, building upon previous knowledge, and delivering the design within four weeks, before the eleventh week of the project, to support the subsequent Proof of Concept development for solving IoT data interoperability issues in iPaaS.

Objective 4
Develop a Proof-of-Concept implementation of the designed iPaaS architecture within five weeks, before the thirteenth week of the project, ensuring it supports Semantic Interoperability for at least three relevant cases for eMagiz, while allocating necessary resources and technical expertise for development to prove its relevance in the case study.

Objective 5
Assess the real-world performance of the proposed solution by applying it to at least three IoT data integration cases in a case study, with the evaluation to be completed within four weeks, before the seventeenth week of the project, focusing on impact, effectiveness, and efficiency.

Objective 6
Evaluate and validate the proposed iPaaS architecture's impact on development efficiency and governance in IoT application integrations by improving IoT integration development time with the help of a Proof-of-Concept case study within four weeks before the twenty-first week of the project.

4.4 Success criteria

4.4.1 Stating success criteria

For stating success criteria, two methods with relation to information systems have been considered. These methods have been retrieved from a review by Castro et al. (2019).

Square Route by Atkinson

The first method is the Square Route by Atkinson (1999). This method consists of four categories, each consisting of different success criteria. This Square Route can be seen in figure 10.

The first category is the Iron Triangle, which is a simple framework consisting of three success criteria: Cost, Quality, and Time. The Iron Triangle is widely used in projects because it provides a simple and effective way to measure project success. In some versions of the Iron Triangle, the criterion of Quality is replaced with Scope.

The second category is The Information System, which consists of four success criteria: Maintainability, Reliability, Validity, and Information Quality Use.

The third category is Organizational Benefits, which consists of six success criteria: Improved Efficiency, Improved Effectiveness, Increased Profits, Strategic Goals, Organizational Learning, and Reduced Waste.

The last category is the Benefits of the Stakeholder Communities, which consists of five success criteria: Satisfied Users, Social and Environmental Impact, Personal Development, Professional Learning, and Contractors Profits.

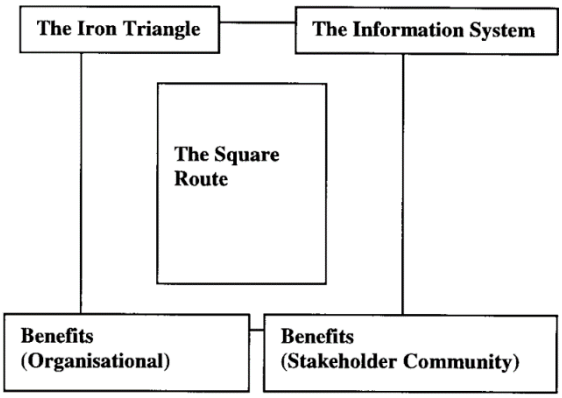

Figure 10: The Square Route by Atkinson (1999)

Success Criteria in IT projects by Thomas & Fernández

The method of Success Criteria in IT projects by Thomas & Fernández (2008) consists of three different categories, each consisting of different success criteria. An overview of this method can be seen in figure 11.

The first category is Project Management, which consists of seven success criteria: On-time, On-budget, Sponsor satisfaction, Steering group satisfaction, Project team satisfaction, Customer/user satisfaction, and Stakeholder satisfaction.

The second category is Technical, which consists of six success criteria: Customer/user satisfaction, Stakeholder satisfaction, System implementation, Met requirements, System quality, and System use.

The last category is Business, which consists of three success criteria: Business continuity, Met business objectives, and Delivery of benefits.

Success criteria	Category		
	Project management	Technical	Business
On-time	X		
On-budget	X		
Sponsor satisfaction	X		
Steering group satisfaction	X		
Project team satisfaction	X		
Customer/user satisfaction	X	X	
Stakeholder satisfaction	X	X	
System implementation		X	
Met requirements		X	
System quality		X	
System use		X	
Business continuity			X
Met business objectives			X
Delivery of benefits			X

Figure 11: Success Criteria in IT Projects by Thomas & Fernández (2008)

To select a method for constructing success criteria, the consideration for each method is discussed. The relevant and less relevant criteria are given in table 1.

Method	Relevant Criteria	Less Relevant Criteria
Square Route by Atkinson (1999)	Time	Cost
	Quality/Scope	Increased Profits
	Maintainability	Reduced Waste
	Reliability	Satisfied Users
	Validity	Contractors Profits
	Information Quality Use	
	Improved Efficiency	
	Improved Effectiveness	
	Strategic Goals	
	Organizational Learning	
	Social and Environmental Impact	
	Personal Development	
	Professional Learning	
Success Criteria in IT projects by Thomas & Fernández (2008)	On-Time	On-Budget
	Sponsor satisfaction	Business Continuity
	Project team satisfaction	Steering group satisfaction
	Stakeholder satisfaction	Customer/user satisfaction
	System Implementation	
	Met requirements	
	System quality	
	System use	
	Met business objectives	

Method	Relevant Criteria	Less Relevant Criteria
	Delivery of benefits	

Table 1: Overview of Relevant Criteria for the given Methods

Both methods do not perfectly match with the context of this project, but a selection of fitting and relevant criteria can be made based on these two methods. The selected criteria are given in table 2.

Criteria	Reasoning
On-Time	The time aspect is relevant for this research project because it has a structured planning. If the time aspect is neglected, the success of the project could be impacted.
Quality/Scope	The quality/scope aspect relevant for having a clear view of the project. It is chosen to implement the Scope element as criterion.
Validity	An important aspect of the research is the validity. To make sure the result and treatment is valid, success criteria based on validity should be implemented.
Met requirements	A success criterion could be having a retrospective to the specified requirements, and measure if they have been met.
Improved Efficiency	Because evaluation of the artifact is an important aspect, the improved efficiency should be one of the success criteria.
Improved Effectiveness	Because evaluation of the artifact is an important aspect, the improved effectiveness should be one of the success criteria.
Delivery of benefits	Because evaluation of the artifact is an important aspect, the delivered benefit should be one of the success criteria.
Personal Development	This is a master research project, so one success criteria should be personal development of the conducting researcher.
Organizational Learning	This master research project is conducted within a company setting. The organization should also learn something from the project.
Stakeholder satisfaction	The involved stakeholders should be satisfied to support the results of the project, and to use the project outcomes.
System Implementation	The artifact will be demonstrated using three different cases, these cases should be implemented successfully.

Table 2: Selected Success Criteria based on the two methods

4.4.2 Stated success criteria

A selection of 11 Success Criteria has been made in the methodology section of this chapter. These success criteria are based on two different methods for stating success criteria within Information System projects. A more detailed description of these methods is given in the methodology section of this chapter.

On-Time

The first success criterion is that the project should be following the planning, and thereby must be handed in on-time. There are a few deadlines within the project planning, which all need to be reached on time to successfully complete the project. The most important deadline is the Green Light for the colloquium. This Green Light needs to be achieved between the 16th week of the project (51st week of the year) and the 20th week of the project (3rd week of the next year). This because it is planned to finish the project at the end of the 23rd week of the project, which means that the Green Light needs to be achieved at least a month before this date. This needs to be at least a month because the University of Twente mentions that the Green Light needs to be achieved at least a month before the colloquium.

The second important deadline is the final deadline of the project. This will be at the end of the 23rd week of the project (6th week of the year). This deadline is for handing in a final version of the research project and uploading it to essay.utwente.nl.

Scope
The second success criterion is that the project should stay within the given scope of the project. When the requirements of a project go beyond those included in the original plans without authorization or control measures, a so-called scope creep occurs (Rudder & Main, 2020). This scope creep has a big impact on the successfulness of the project, and therefore needs to be avoided. According to Rudder & Main, this can be done by actions like stating clear requirements, using a Gantt-chart, and getting approval from stakeholders on the scope. The scope of this project is given in chapter 3.

Validity
The third success criterion is that the research project should be valid. The validity will be measured in the Evaluation stage of this project. The validity of the research will be demonstrated with the help of three different cases. Further, the proposed solution will also be evaluated. If the project is not valid, the results of the project can be seen as not successful.

Met requirements
The fourth success criterion is that the stated requirements should be met. The requirements are an important factor in designing and developing the proposed solution. The requirements are prioritized following the MoSCoW method, which divides the requirements in four different categories. At least all the requirements with the Must Have classification should be met to make this project a success.

Improved Efficiency
The fifth success criterion is the improved efficiency of the proposed solution. Efficiency refers to “doing things right” or “doing more with less” (Schwarz, 2022). The Proof of Concept should improve the efficiency of the current IoT data integration process in iPaaS, with the help of Semantic Interoperability. If the efficiency is not improved, this project can be seen as not successful.

Improved Effectiveness
The sixth success criterion is the improved effectiveness of the proposed solution. Effectiveness refers to “doing right things” or “the quality that resources deliver” (Schwarz, 2022). The Proof of Concept should effectively incorporate Semantic Interoperability in iPaaS for IoT data integrations. If this is not effective, this project can be seen as not successful.

Delivery of benefits
The seventh success criterion is that the project should deliver benefits to the stakeholders. The benefits in this case can be improved efficiency, effectiveness, governance, or knowledge. If the project does not deliver benefits to at least one important stakeholder, the project will not be a success.

Personal Development
The eighth success criterion involves personal development. Maslow (1970) suggested that all individuals have the in-built need for personal development which occurs through the process of self-actualisation as mentioned in his famous pyramid of needs. In the context of this study, the project should facilitate the personal development of the conducting researcher. This could be done through acquiring new skills, gaining valuable experiences, or expanding knowledge. If personal development is not realized, the project may be considered less successful.

Organizational Learning
The ninth success criterion involves organizational learning. This criterion emphasizes the importance of the acquired knowledge and insights for the organization where this study is conducted. If the study does not provide learning for the organization, it can be seen as not successful.

Stakeholder satisfaction
The tenth success criterion is the satisfaction of the stakeholders. This involves managing the stakeholders based on their power and interest. If the important stakeholders are not satisfied with the project, the successfulness could be impacted. To ensure that the important stakeholders are satisfied, a strategy for managing stakeholders should be followed.

System Implementation
The eleventh success criterion is the implementation of the system. The system is in this case the developed Proof of Concept, the implementation refers to the different use cases and scenarios to validate the feasibility and functionality of the proposed solution. This criterion looks at the operational reality, demonstrating its practicality and effectiveness in addressing the identified requirements. If the system implementation does not follow the plans, the project will not be a success.

4.5 Conclusions
Different existing solutions for iPaaS have been identified. Market leaders like Oracle and MuleSoft have a clear vision of the future of iPaaS, in combination with the ability to execute this vision. The iPaaS platform which will be used in this project, eMagiz, can be seen as a niche player based on its location and use cases.

The stakeholders within this project have been analysed and have been plotted onto the stakeholder matrix. Important stakeholders are Integration Specialists, Product Owners, and Supervisors. These stakeholders need to be closely managed. Further, the Platform Developers need to be kept satisfied, while the End Users need to be monitored with less effort.

Further, objectives have been stated. These objectives involve the set of goals for the execution and results of this project. Next to the goals, also different success criteria have been stated. These criteria should be met to make the research project a success.

5. Results

In this chapter, the different results are discussed. This is part of the third stage as mentioned in the DSRM by Peffers et al. (2007). The first two research questions stated below will be answered with the help of a literature study.

> (SQ1) - How can IoT data integrations be incorporated in iPaaS architectures?
>
> (SQ2) - How can Semantic Interoperability be incorporated in iPaaS architectures?

Based on the outcomes of this literature study, a solution design will be created. This solution design will be the answer to the third research question.

> (SQ3) - What are the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across different systems?

First, the qualitative findings for the first two research questions will be stated. Next, an overview of the requirements to the treatment will be presented and prioritized. Finally, a solution design based on these findings and requirements will be given.

5.1 Qualitative Findings

5.1.1 Research Methodology

5.1.1.1 Desk Research

For finding relevant literature to answer the research questions, desk research will be conducted. This desk research will be conducted with the help of Google Scholar based on different relevant keywords. The results of Google Scholar are based on the relevance of the search prompt. To ensure the actualization of the results, the results will be filtered based on their publication date. After gathering the resources, the resources will be uploaded in Mendeley, which is a reference tool for Academics. This tool is provided by Elsevier and is accessible with an Utwente Institution account.

The resources in Mendeley will be selected based on their content to ensure the relevance to the project. With these selected resources, a valuable foundation for the design phase of this project can be made.

First, a further understanding of iPaaS architectures is needed, this is done with the help of the keywords “iPaaS architecture”, or “integration platform architecture”. This did not result in the desired results. Therefore, it was chosen to include one Gray Literature source. This source is a website giving a detailed description of an iPaaS architecture.

Next, to answer SQ1, the keywords “IoT Integration Platform Architecture” and “Implementation of IoT [Protocol]” will be used. This [Protocol] refers to a set of the most used protocols for IoT communication.

Finally, for answering SQ2, the keywords “Semantic Interoperability Architecture”, “Semantic Interoperability IoT Implementation”, and “Semantic Interoperability IoT solution” will be used.

With the answers on these questions, an artifact can be designed. This artifact will be the answer to the third research question, SQ3.

5.1.1.2 Architecture Framework

There are different languages for describing architectures. Ivanovich (2019) mentioned five Architecture Description Languages (ADL). These ADLs are described below. The different ADL’s have been considered, and it was chosen to use the ArchiMate ADL based on its usability.

The ArchiMate ADL was introduced by Jonkers in 2004. ArchiMate is a graphic description of both information, corporate, and engineering systems. The ArchiMate ADL has been transferred to The Open Group, where it is slated to become the standard for architectural description accompanying The Open Group Architecture Framework (TOGAF) (Lankhorst et al., 2009). The Open Group is a global group enabling the achievement of business objectives through technology standards. TOGAF is a standard approach for assisting in the acceptance, production, use, and maintenance of Enterprise Architectures governed by The Open Group (Josey, 2018).

The core framework of ArchiMate follows a three-layer model, which enables linking high-level descriptions. The Business layer illustrates the business services that are offered to the customers. These services are being realized in organizations by the business processes performed by the business actors. The Application layer is a supporting factor for the business layer. The application services in this layer are realized by the applications components. The Infrastructure layer, also called Technology layer, consists of the infrastructure services like processing, storage, or communication, which are needed to run the applications. These services are realized by hardware and system software (Meertens, 2012).

The standard of ArchiMate consists of six main components: a framework, abstract syntax, modelling concepts, language semantics, concrete syntax in terms of a visual notation, and a viewpoint mechanism (Lankhorst et al., 2009).

ArchiMate uses a graphical representation of the architecture, open documentation, and the support of various tools, which makes it suitable for educating bachelors and master students in the field of IT (Ivanovich, 2019). ArchiMate is easy to understand, providing the free Archi tool for modelling architectures with this language. An example architecture of a Website following the ArchiMate Language can be seen in figure 12.

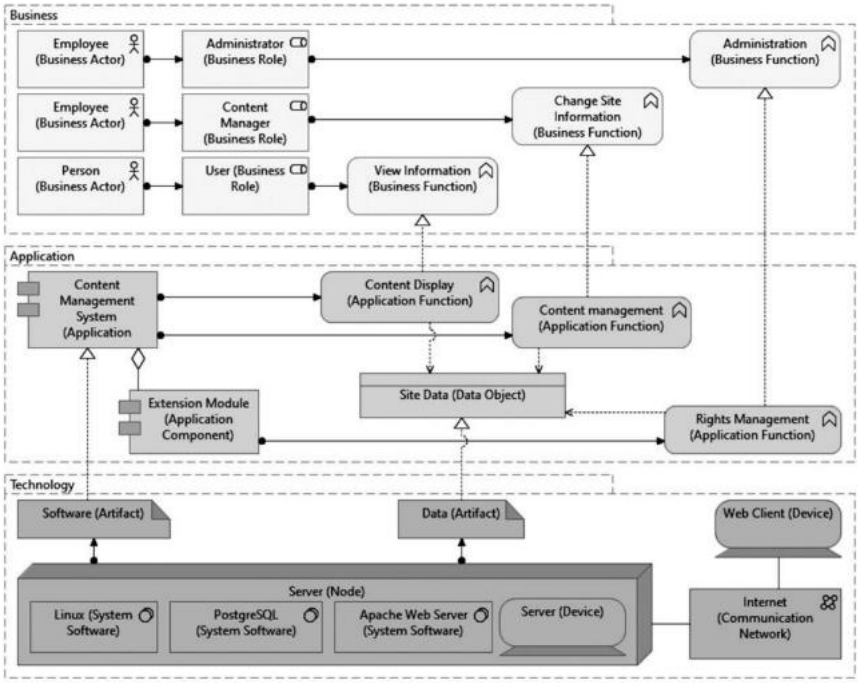

Figure 12: Example Architecture of a Website following the ArchiMate Language by Ivanovich (2019)

The ArchiMate ADL was designed to describe an architecture in a way that makes it understandable to most stakeholders and should be used to create an overview of an architecture with just enough

details (Sarrodie, 2018). As Ivanovich (2019) mentioned, ArchiMate is a relevant language for bachelor and master studies in the field of Information Technology. Because ArchiMate provides a holistic view, ArchiMate will be used as ADL in this project.

5.1.2 IoT data integrations in iPaaS architectures

This chapter of the study seeks to answer the first research question: "How can IoT data integrations be incorporated into iPaaS architectures?" To address this, first an exploration is given of various iPaaS architectures. This is followed by an explanation of different IoT architectures. Finally, it is explained how these IoT integrations can be seamlessly incorporated into iPaaS architectures.

To better understand the stated architectures, the ArchiMate Architecture Description Language is used. The motivation and description of this method is given in the methodology section of this chapter. Based on the acquired knowledge, a general architecture is created made with the help of the ArchiMate language.

5.1.2.1 Architecture of iPaaS

To design an iPaaS architecture for supporting IoT data integrations, first the general idea of iPaaS architectures needs to be explored. There is limited literature on the architecture of integration platforms, let alone the architecture of iPaaS. Because of these limited resources, it is chosen to include limited grey literature for understanding these architectures. Grey literature involves information produced outside of traditional publishing and distribution channels (SFU Library, sd).

Below, different found iPaaS architectures are described. These architectures lead to a general iPaaS architecture which can be used to answer the first two stated research questions.

Reference Architecture for Integration Platforms by Singh, Van Sinderen & Wieringa (2017)
The first architecture can be seen in figure 13. This architecture is the Architecture for Integration Platforms by Singh, Van Sinderen & Wieringa (2017), and was developed to help enterprises make better design and solution choices. This should benefit both practitioners and academics in the field of integration platform development and analysis. This Reference Architecture (RA) consists of three different layers. The data layer, the application layer, and the service layer.

The first layer as described by Singh, Van Sinderen & Wieringa (2017) is the data layer. This layer uses adapters to connect to the different data sources. Each source requires its own dedicated adapter due to the interoperability problems. The Data Bus acts as a common connection point for the data sources at the device level. This allows efficient data collection from multiple devices. The Data Filtering filters the data based on its intended use and removes irrelevancies or errors in data. This is done with rules and policies that can be adjusted accordingly. The Meta-data, which is data about the data, enhances the organization and management of the data within the integration platform. The Data Conversion & Common data model is used to facilitate Semantic Interoperability between applications by providing standardized representations for the data from different sources, while a Data Conversion module could be used to make additional data transformations. The database provides the storage for the data from the different sources. The database could also be used to store the local data of an integration platform like a list of devices to connect to, rules for data filtering, or meta-data. The Enterprise Service Bus (ESB) is used to enable applications to access the data from the database. An ESB acts as a central connection point for all applications in an enterprise.

The second layer as described by Singh, Van Sinderen & Wieringa (2017) is the application layer. This layer uses an Event Processing (EP) module to make decisions based on the incoming data. This EP determines the context and detects events based on business rules, models, algorithms, and ontologies to make decisions. The Workflow Management Module handles the sequence of activities

which are triggered by the events. This can involve applications, human interactions, or data requests. The business rules are enterprise policies that describe actions when specific events occur within the workflow. The Data Source Manager is responsible for managing the connected data sources. This ensures easy updates, repairs, and diagnostics. The Analytics and Decisions Support analyse contextual data and can perform various mathematical operations to support the making of decisions. The Computation performs data computations and presents results through a dashboard, this can trigger different processes and procedures based on these results. The Diagnostics module monitors the overall platform performance, the devices, and the applications. The Platform Manager utilizes the diagnostics module and allows admins to view, edit, or update the connected data sources.

The last layer as described by Singh, Van Sinderen & Wieringa (2017) is the service layer. This layer involves a dashboard which provides output from various applications to the end users. Next, the Platform Management Tool (PMT), is a graphical user interface (GUI) for managing the platform. This can be used by the admins to perform administrative functions on the platform. The Web Service Module (WSM) finally allows the external applications to access the services provided by the applications of the integration platform.

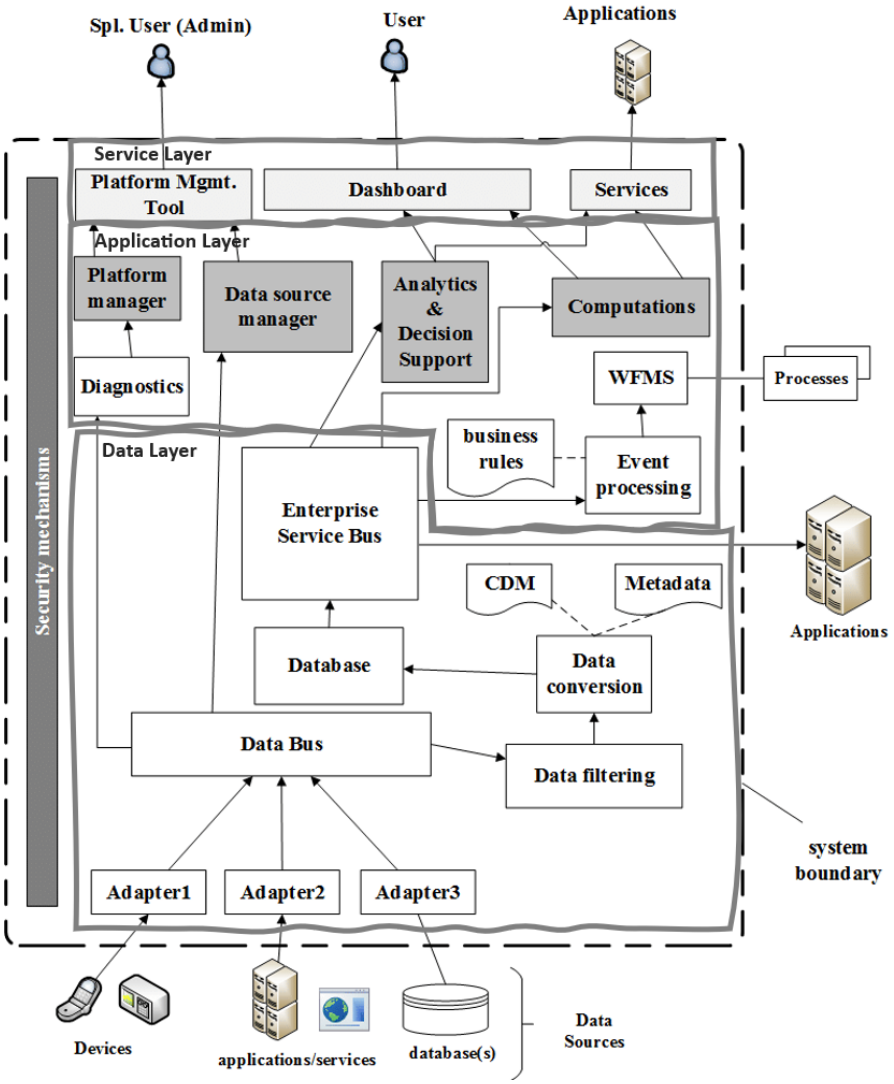

Figure 13: Reference Architecture for Integration Platforms by Singh, Van Sinderen & Wieringa (2017)

Information Integration Platform Architecture by Xu et al. (2020)
The second presented architecture is the Information Integration Platform Architecture by Xu et al. (2020), which can be seen in figure 14. The integration platform itself consists of three layers: Data Collection, Core Service, and Presentation. The first layer, the Data Collection layer, directly communicates with the different communication network systems. This layer collects the information of these systems via the HTTP and MQTT protocol as raw data, which is then pushed to the Data Fusion Service of the core service layer.

The second layer, the Core Service layer, acts as the hearth of the platform with the help of different services. The Data Fusion Service combines the raw data collected by the Data Collection Layer to create situational information as a basis for the other services. The Workflow Execution Service incorporates a dynamic workflow model to enable workflows, monitoring, and data exchange with the other included services. The Data Push Service sends the various types of data to the Presentation layer based on the data binding principles and workflows.

The third layer, the Presentation layer (also called the 'Comprehensive' Presentation layer), involves all the different kinds of user interfaces. These user interfaces consist of topology, situations, fault alarms, workflows, task planning, resources, equipment inspection, and performance monitoring.

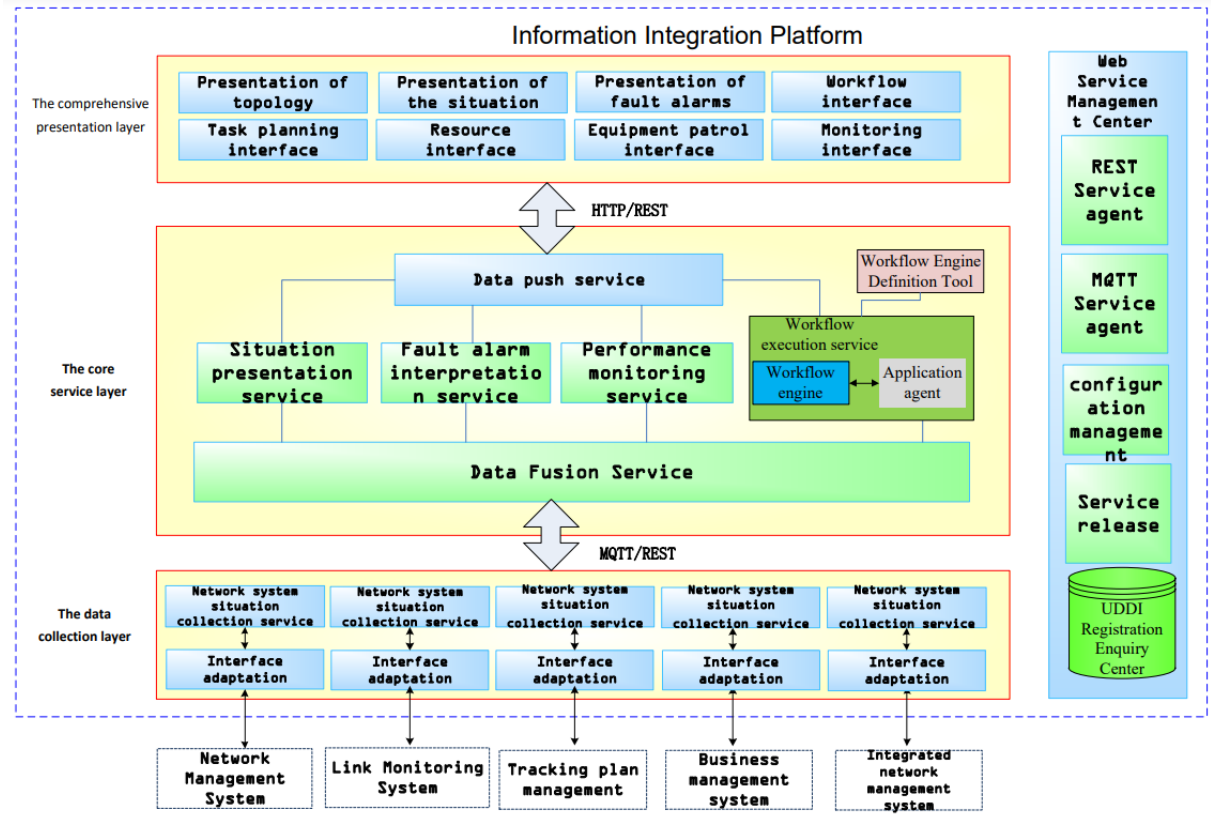

Figure 14: The Information Integration Platform Architecture by Xu et al. (2020)

According to by Xu et al. (2020), this platform architecture is reliable, scalable, independent, and interoperable. The interesting component in this architecture is the Data Fusion Service. This service combines all the incoming data into a data collection, which triggers a workflow, which next pushes the data to the interfaces.

MuleSoft's Integration Architecture by Schneider & Abeck (2023)
Schneider & Abeck (2023) presented a comparison of Service Oriented Architectures (SOA) and the

Architecture of the MuleSoft iPaaS. MuleSoft is, as described in the Existing Solutions chapter of this research, the iPaaS of Salesforce, combining API management, integration, and automation in one MuleSoft Anypoint Platform. The architecture of MuleSoft consists of five layers and can be seen in figure 15. The first layer is the Presentation layer. This layer contains all the user interfaces, and can be compared with the Service layer in the RA of Singh, Van Sinderen & Wieringa (2017), and the Presentation layer in the Information Integration Platform by Xu et al. (2020).

The second layer is the Experience Layer. This layer reconfigures the data, so that it can be easily consumed by its intended audience (MuleSoft, 2022). The data all comes from a common data source, instead of separate point to point integrations. In most cases, many applications want to use the data, but they handle different formats. This data is transformed by the Experience API, so it can be consumed in the right format.

The third layer is the Process layer. This layer involves the underlying business processes which interact with and reshape the form of the data. This process layer should be independent of the source and target systems. This layer involves the so-called Process API, which performs specific functions and provides access to the data.

The fourth layer is the System layer. This layer involves the core systems of records. These systems are often not easily accessible because of connectivity concerns. This layer consists of the System API, which enables accessing the underlying system of records in a canonical format. This System API exposes the data which is obtained out of the system of records.

The fifth layer is the Data layer. This layer contains all the data sources, and can be compared with the Data layer in the RA of Singh, Van Sinderen & Wieringa (2017), and the Data Collection layer in the Information Integration Platform by Xu et al. (2020).

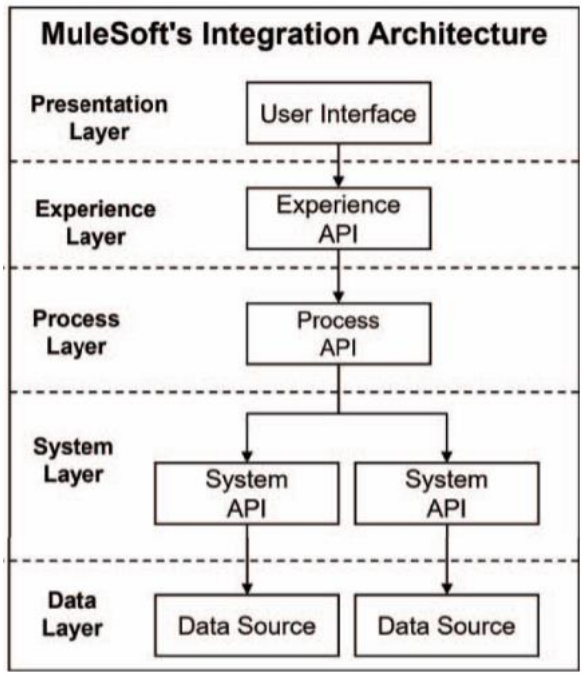

Figure 15: MuleSoft's Integration Architecture by Schneider & Abeck (2023)

iPaaS Reference Model by Paradkar (2018)
Paradkar (2018) also provided a RA for iPaaS, consisting of eight layers as seen in figure 16. The first layer is the IDE & Developer Productivity layer. This layer enables the users to model, develop, configure, test, and deploy the integration flows with the help of an Integrated Development

Environment (IDE). This IDE can be described as an integrated, graphically oriented modelling environment which makes it possible for the developers to model the integration flows (Paradkar, 2018).

The second layer is the Business Process Management (BPM) layer. This layer provides the user with the ability to execute the integration flows, which have been developed in the IDE. These workflows are based on the Workflow Definition Language (WDL).

The third layer is the Enterprise Service Bus (ESB) & Application Integration layer. This layer facilitates the integration of multiple applications on a bus-like architecture. The ESB provides the integration of applications by a communication bus between these applications. This bus decouples the applications from each other, which allows communication without the dependency or knowledge of the other application involved on the bus.

The fourth layer is the Business-to-Business (B2B) Integration layer. This layer facilitates the integration of data from different back-end systems from external applications. B2B integrations consist of the transport, formats, protocols, validation, adapters, security, and B2B partners.

The fifth layer is the Data Integration layer. This layer enables the aggregation of data form the different data sources into an integrated view. The capability of this layer involves bulk/batch delivery of data to integrate data from databases, execute queries on the different data sources, and synchronizing data between the different databases. This layer also involves the mapping, transformation, and enrichment of the data.

The sixth layer is the Application Programming Interface (API) Management layer, which also involves the Microservices. This layer involves the governance of APIs, which can be public APIs, private APIs, or APIs from partners. The APIs provide a critical capability of the platform, which needs the governance, performance, and security to ensure the delivery of the business outcomes.

The seventh layer is the Governance & Security layer. To support the governance processes which are essential in an iPaaS, governance platform services need to be present. These services support users to agree on, define, and apply policies for enforcing governance decisions which come with endpoint interfaces. Further, this layer also supports the compliance with the policies in the form of metrics, and the support of the decision on how to handle the exceptions.

The eighth layer is the Monitoring & Management layer. This layer enables users to deploy and administrate integration flows, monitor the execution of these flows, manage the behaviour of these flows, and perform other tasks for monitoring and managing the platform. With these services, the layer provides the capabilities to monitor, manage, and administrate the runtime execution of the integration flows.

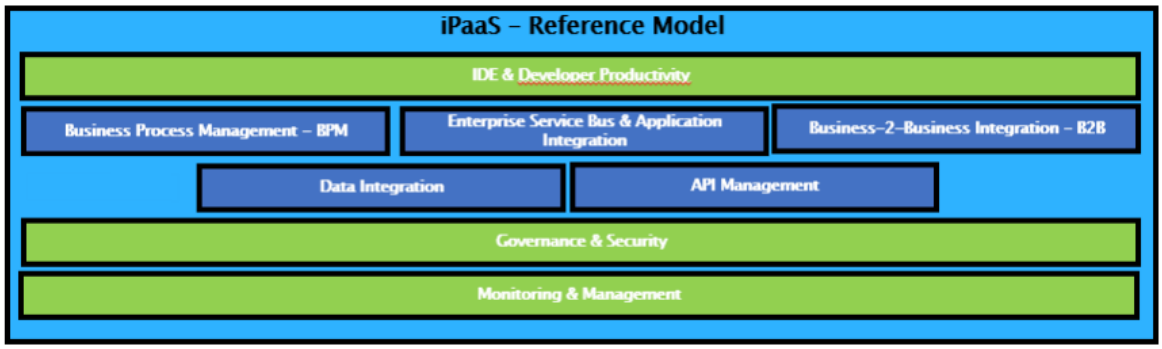

Figure 16: iPaaS Reference Model by Paradkar (2018)

This reference model provides a clear structure of eight different layers of which an iPaaS architecture could consist of. This reference model is less technical than other provided solutions, but it does give a clear overview of the essential capabilities of an iPaaS. It was chosen to also describe this model, because it is one of the few sources especially describing iPaaS in the form of an architectural model.

Enterprise iPaaS Reference Architecture by Abeysinghe (2021)
Abeysinghe (2021) provided a RA for enterprise iPaaS. This RA consists of different planes as seen in figure 17. The first plane is the Control-plane. This plane consists of different portals with the focus on the full API lifecycle management, high-code/low-code/no-code-driven integrations, business intelligence (BI), API marketplace, identity & access management (IAM), quality assurance, governance, and observability (Abeysinghe, 2021). With this control plane, policies, configurations, and integration logic can be defined.

The Full API lifecycle Management / Security component helps API creators to develop, document, scale, and version their APIs, while facilitating the related tasks like publishing, monetising, and promoting. The Security part manages the tokens, protects APIs, and defines the security policies.

The API Marketplace helps building an API ecosystem with the help of multiple parties listing and offering their APIs in a type of store. This can be called Business to Business to Consumer (B2B2C). This marketplace enables organizations to become more agile and competitive.

The high-code/low-code/no-code component enables the integrations specialists to create the integrations for their organizations. The high-code integrations involve the developers to write and deploy code, which requires experienced specialist to develop these integrations. The low-code integrations eliminate the main coding aspects with the help of a drag-and-drop interface, although sometimes some coding is still required. This can be referred to as template-driven integrations and covers a wide number of use cases with standard connectors. The no-code integrations provide out-of-the-box templates for integrating applications with drag-and-drop interfaces. No-code covers all the standard less-advanced integration patters matching the templates.

The Data-plane and the Control-plane are connected through the Continuous Integration & Continuous Delivery (CI/CD) pipeline. This CI/CD pipeline is essential to accelerate the innovation with the feedback of customers. This pipeline typically consists of one of more steps. The first step turns the source code into a build, which is tested and deployed across the different environments until they reach the end-users.

The next plane is the Data-plane, which is the place where the defined policies, configurations, and integration logic are used, with the help of proxies and gateways for APIs, integrations, streaming, workflows, and messaging. This plane also captures the metrics, logs, and data tracing to get business insights reports about the behaviour of the entire system. These reports are critical for the decisions made in the business.

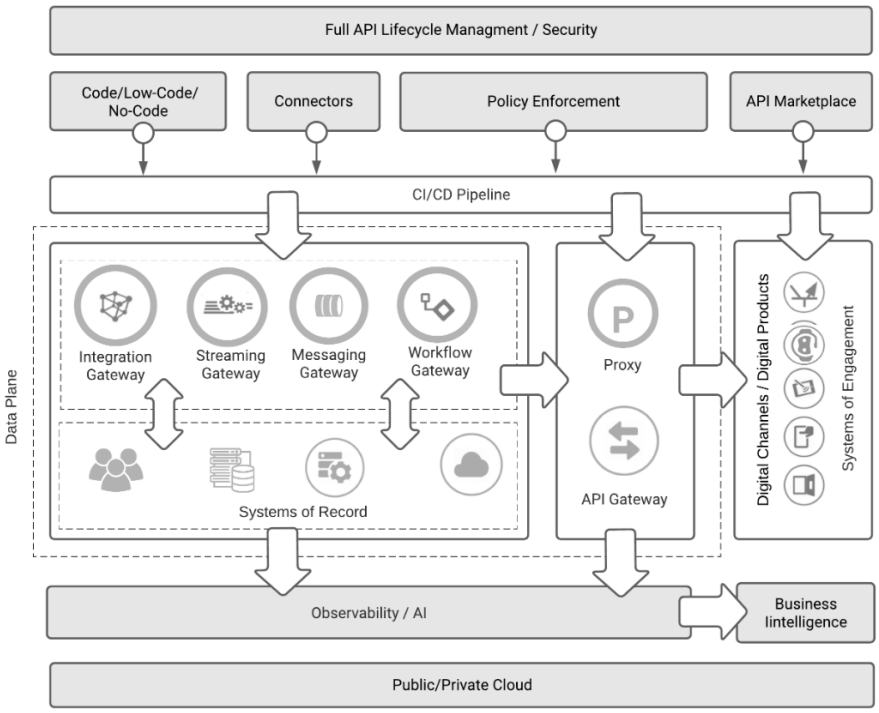

Figure 17: Enterprise iPaaS Reference Architecture by Abeysinghe (2021)

This RA for iPaaS seems complete, but as mentioned before, the resource is “Grey literature”. Abeysinghe created this framework for WSO2, which is an open-source technology provider, also providing an iPaaS.

Conclusion

Most of the stated architectures in iPaaS / Integration Platforms consist of a clear layered model. The topmost layer in almost all these layers can be described as the Presentation layer. This layer presents the Integration Platform in an intuitive interface. In this interface, the platform can be managed, integrations can be created, and flows can be configured. In the RA for Integration Platforms by Singh, Van Sinderen & Wieringa (2017), this layer is called the Service layer, which involves the Dashboard, Platform Management Tool, and the Services. In the Information Platform Architecture of Xu et al. (2020), this layer is called the Comprehensive Presentation layer, which consists of all kinds of presentations and interfaces. In the MuleSoft Integration Architecture by Schneider & Abeck (2023), this layer is called the Presentation layer, which involves the User Interface. In the iPaaS Reference Model by Paradkar (2018), this layer could be seen as a combination of the Integrated Development Environment & Productivity layer, and the Monitoring & Management layer. Finally, in the Enterprise iPaaS RA by Abeysinghe (2021), a clear name to this layer has not been stated. But based on the given components, the top two sets of components can be seen as a layer presenting the iPaaS. These components consist of Code/Low-Code/No-Code, Connectors, Police Enforcements, and an API Marketplace.

The next layer can be seen is the Application layer. This layer enables the management of the different components, combined with the execution of processes and diagnostics of the platform. In the RA for Integration Platforms by Singh, Van Sinderen & Wieringa (2017), this layer is called the Application layer, which involves components like the Platform Manager, Data Source Manager, Analytics & Decision Support, and Computations. In the Information Platform Architecture of Xu et al. (2020), this layer can be seen as the Core Service layer, which involves the Monitoring service, and

the Workflow Execution service. The Data Fusion & the Data Push services would not belong in the Application layer, but rather in the Data layer described below. Further, the Web Service Management Centre would also be included in the Application layer. In the MuleSoft Integration Architecture by Schneider & Abeck (2023), this layer can be seen as a combination of the Process and System layers. These layers enable the extraction of the Canonical Data Model and the execution of the integration processes. In the iPaaS Reference Model by Paradkar (2018), this layer can be seen as a combination of the Business Process Management (BPM), and the different integration components. Finally, in the Enterprise iPaaS RA by Abeysinghe (2021), a clear name for this layer has not been stated, but a selection of the components can be made. These components are the Observability / AI, Business Intelligence, Cloud, and Systems of Engagement components.

The next layer can be seen is the Data layer. This layer involves the receiving, transformation, and routing of the data. In the RA for Integration Platforms by Singh, Van Sinderen & Wieringa (2017), this layer is called the Data layer, which involves components like the Enterprise Service Bus, Data Base, Conversions, and Adapters. In the Information Platform Architecture of Xu et al. (2020), this layer can be seen as the Data Collection layer, which involves the collecting of the data with the help of the Interface Adapters. Further, the Data Fusion Service, and the Data Push Service from the Core Service layer of this architecture also could belong to this Data layer. In the MuleSoft Integration Architecture by Schneider & Abeck (2023), this layer can be seen as a combination of the Experience and Data layers. These layers enable the extraction of data from the sources, and the reconfiguration of data for easy consumption. In the iPaaS Reference Model by Paradkar (2018), this layer is not present. The model involves the integration of data, which is already mapped onto the Application layer, but does not explicitly mention this in a specific component. In the Enterprise iPaaS RA by Abeysinghe (2021), a clear name to this layer has not been stated, but a selection of the components can be made from the Data Plane. These components involve the Proxy & Gateways, combined with the CI/CD pipeline and the System of Records.

Additionally, the RA for Integration Platforms by Singh, Van Sinderen & Wieringa (2017), the iPaaS Reference Model by Paradkar (2018), and the Enterprise iPaaS RA by Abeysinghe (2021) all involve a Security component, which is missing in the representation of the other architectures. To get a clear architecture of iPaaS, this Security mechanism needs to be involved as well. A general iPaaS architecture can consists of three horizontal layers, and one vertical layer. This architecture follows the layers as described in the RA by Singh, Van Sinderen & Wieringa (2017). An illustration of the layers of this iPaaS architecture is created and shown in figure 18.

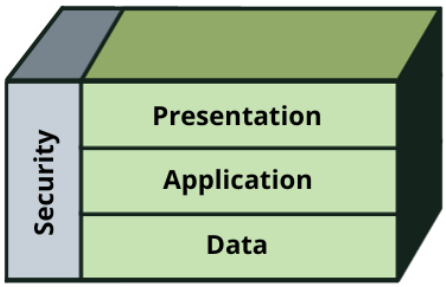

Figure 18: Illustration of General iPaaS Architecture Layers

A general overview of an iPaaS architecture, following the ArchiMate ADL, is given in figure 19. In this architecture, different components of the described architectures have been used. The core of the architecture is based on the RA for Integration Platforms by Singh, Van Sinderen & Wieringa (2017). The Workflow Execution Service and the Workflow Engine Definition tool, which is interpreted as the Workflow Definer, have been adopted from the Information Platform Architecture of Xu et al. (2020).

Further, the Low-Code aspect, and the Connector Marketplace, have been adopted from the Enterprise iPaaS RA by Abeysinghe (2021). The components of the MuleSoft Integration Architecture by Schneider & Abeck (2023) and the iPaaS Reference Model by Paradkar (2018) are already represented by the different components of RA by Singh, Van Sinderen & Wieringa (2017).

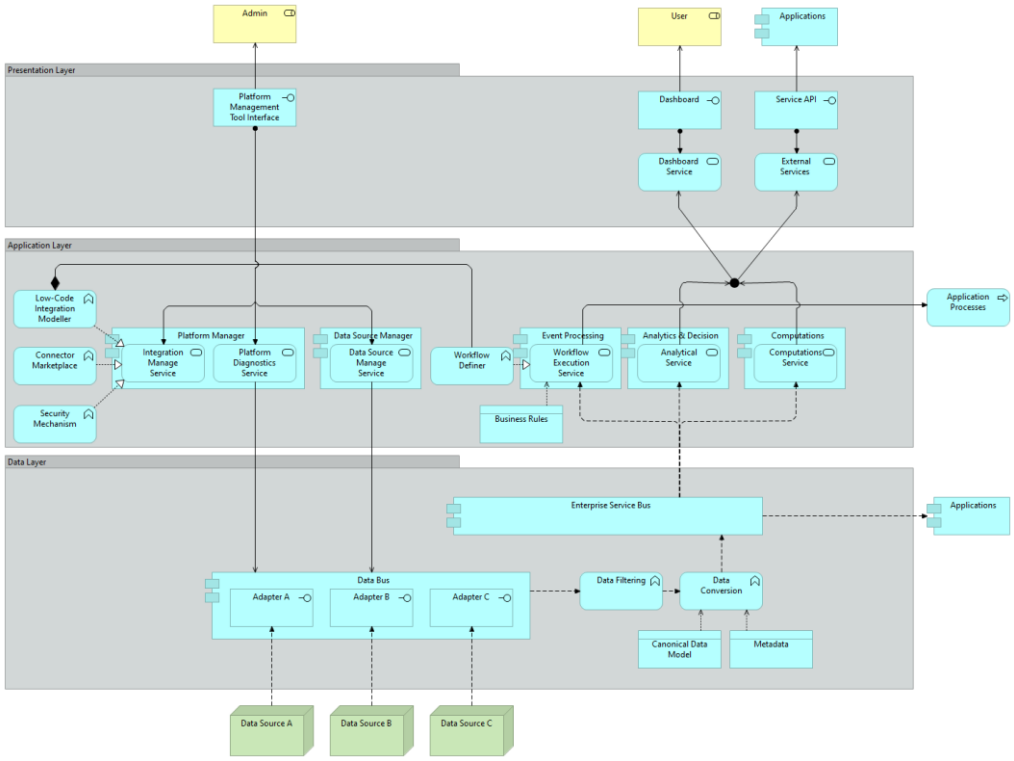

Figure 19: Simplified iPaaS architecture following the ArchiMate ADL

5.1.2.2 Architecture of IoT

Now that a general architecture for iPaaS is known, an investigation into how to incorporate IoT integrations needs to be conducted. Below, different found architectures in the IoT are described. These architectures lead to a general IoT architecture which is finally related to found iPaaS architecture.

Reference Architecture for IoT by Sobin (2020)
Sobin (2020) presented a Reference Architecture (RA) for IoT, which can be seen in figure 20. This RA is based on a white paper by Fremantle (2016). The RA consists of five horizontal layers and two vertical layers.

The first horizontal layer is the Client layer. This layer consists of a web portal, a dashboard, and an API management module. This layer enables devices to communicate outside of the system. The web portal enables the interaction with the devices and the layer below. The dashboard visualises the analytics of the devices. The APIs enable interactions with external systems through programming interfaces.

The second horizontal layer is the Event processing and Analytics layer. This layer processes the events which are provided by the Aggregation/Bus layer, and executes actions based on these events. This layer stores the data in a database, enabling analytical features.

The third horizontal layer is the Aggregation/Bus layer. This layer enables the integration of the different devices into the layers above. This layer needs to involve a HTTP server and or an MQTT

server to enable communication with the devices. This layer also needs to aggregate and combine the communications for the different devices and route these communications to a final device. Further, this layer needs to be able to transform the different protocols and formats of the messages into another desired format. An additional function of this layer could be the mapping of relational models. Security wise, this layer needs to be able to validate the request based on credentials and needs to enforce policies on the incoming requests.

The fourth horizontal layer is the Communication layer. This layer enables the connectivity of the devices. These protocols, and how they can be implemented, are later discussed in this chapter.

The fifth layer is the Device layer. This layer consists of the IoT devices. They can be connected directly to the Communication layer or can be connected via a gateway.

The vertical layers operate with the different horizontal layers. The first vertical layer is the Device Manager layer. This layer incorporates management of the devices, involving the deployment of software, locking, or wiping of the device. This layer also manages a list of device identifiers and its owners.

The second vertical layer is the Identity and Access Management. This layer provides the token issuing and validation, policy management, and the directory of users.

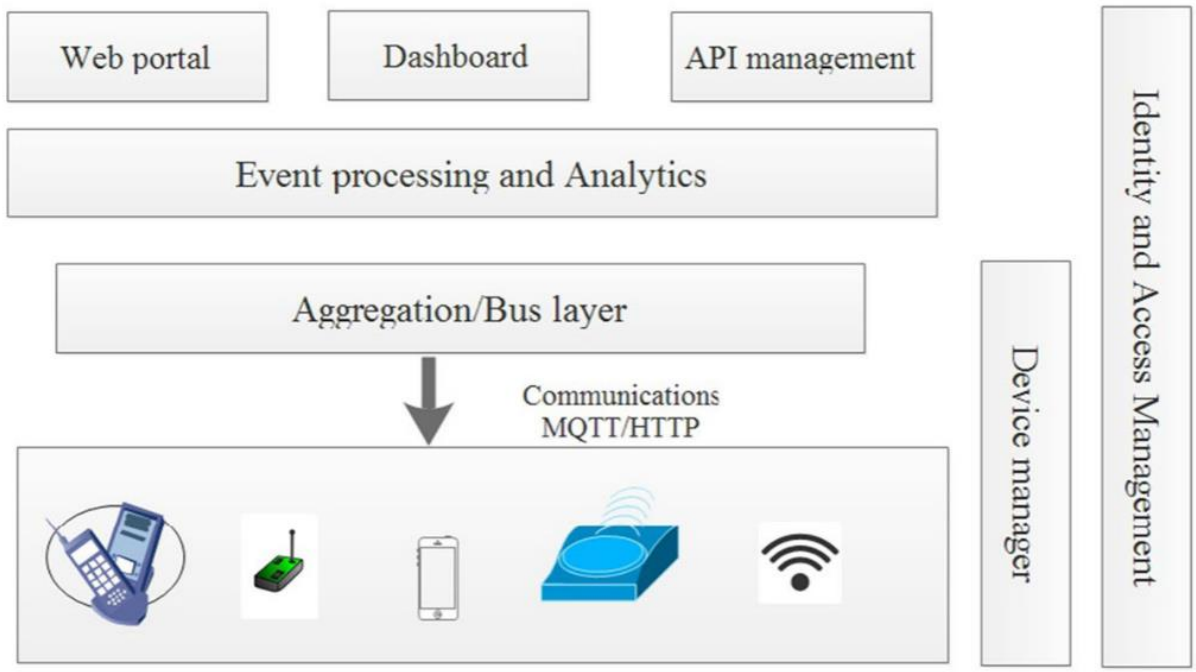

Figure 20: Reference Architecture for IoT as mentioned by Sobin (2020)

The role of iPaaS in this RA for IoT could be on the Aggregation/Bus layer. The iPaaS will support the application protocols to connect with the devices/gateways. The iPaaS will also support the function to transform the messages from different protocols/formats to the desired format.

IoT Reference Architecture by Guth et al. (2016)
Guth et al. (2016) gave a simplified general IoT RA based on four popular architectures, even though the article stems from 2016, it still can be seen as relevant. This RA consist of five general layers, which can be seen in figure 21. The top layer is the Application layer, which represents the software which uses the IoT devices. The software in this layer could be any IoT Application.

The second layer is the IoT Integration Middleware layer. This layer receives the data from the devices, processes it by executing rule-based actions, and sends the commands/data to the Application layer. According to Guth et al., this layer should support appropriate communication technologies like WiFi, with the corresponding communication protocols like HTTP and MQTT. This layer can communicate directly with devices, or via a gateway, making this a layer for integrating different Sensors, Actuators, Devices, and Applications.

The third layer is the Gateway layer. This layer connects devices which cannot directly communicate with the Integration layer. This Gateway layer bridges the limitation by providing support for different protocols and technologies. These protocols can be different communication protocols like Zigbee, BLE, or LPWANs.

The fourth layer is the Device layer. This layer involves the hardware which is connected to the circuit of the Sensors or Actuators. To make a device in the Sensor/ Actuator layer able to process the data, a driver is required. The step from Sensor or Actuators to Drivers represents the switch from the physical to the digital world. Guth et al. (2016) gave two types of devices. Self-Contained devices involve black box functionalities. An example of a Self-Contained device can be a device with an integrated leakage sensor and a water flow valve actuator. This device will close the valve when a leakage is detected. On the other side, System Connected devices require the interference from an application via the Integration Layer.

The fifth layer consists of two sublayers. The first sublayer is the Actuator sublayer. This sublayer involves the hardware components which can "Act upon, Control, or Manipulate the physical environment by giving an optic or acoustic signal" Guth et al. (2016). These Actuators receive commands from the connected device and translate these commands to a physical action.

The second sublayer is the Sensor sublayer. This sublayer involves the hardware components which can "Measure parameters of the physical environment and translate it to an electronic signal" Guth et al. (2016). These sensors can be connected to, or integrated into, a Device from the layer above.

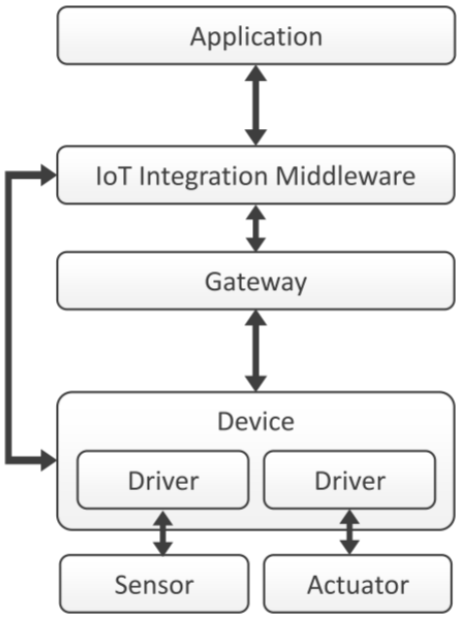

Figure 21: IoT Reference Architecture by Guth et al. (2016)

The role of iPaaS in this RA for IoT could be on the IoT Integration Middleware layer. The iPaaS will support the communication to connect with the Device layer or the Gateway layer. The iPaaS will also

support the function to transform the messages from different protocols/formats to the desired format.

Fog-Enabled IoT platform Architecture by Asemani et al. (2019)
Asemani et al. (2019) provided a comprehensive architecture for fog-enabled IoT platforms. This architecture can be seen in figure 22. The architecture is designed to enable Fog computing in IoT platforms. Laghari et al. (2021) stated a that Fog computing extends cloud computing to the edge of the network, enabling low-latency, context-aware, and distributed applications with three main components: end devices, fog nodes, and back-end cloud infrastructures.

The edge in this case refers to Edge Computing. This Edge Computing has emerged as a promising solution in the field of IoT integrations. This approach positions computing resources closer to devices, instead of centralizing them in massive datacenters.

In contrast with the other two described architectures for IoT, this architecture gives a more comprehensive explanation of the platform layer. The Application layer and the Sensor & Actuator layer, also called Edge layer in this architecture, refers to the same layers as the Device layer of Sobin (2020) and to the Sensor and Actuator layer of Guth et al. (2016).

The interesting layer in this architecture is the Integration Unit from the Cloud IoT Platform layer, also referred to as the Platform on Cloud layer. The Cloud IoT Platform Layer consists of several distinct units and sublayers. In this research, the Integration Unit will be highlighted. This unit consists of the Cloud Integration layer, Platform Integration layer, Data Integration layer, Business system integration, and 3rd Party Services Integration layer.

The Connection and Device Management Unit manages the connections, APIs, and Devices within the platform. This unit communicates with the Cloud Infrastructure which hosts the Computational, Networking, and Storage components of the platform.

The Development Tools Unit enables the creation of IoT integrations on the platform with the help of Simulation tools, cloud & data services interfaces, development & deployment environments, and API management functionalities.

The Service Management Unit enables the management and control of the platform, and the integration and orchestration of the services 3rd party services.

The Data Management & Processing Unit enables the data Management & Storage, analytics of the Data & Events, and the Visualization of the analytics.

Through all these units, the security layer is an important factor, ensuring that the platform is only accessible by authorized and authenticated users and devices. This security is done with components like a firewall, access management, data encryption, and role-based access.

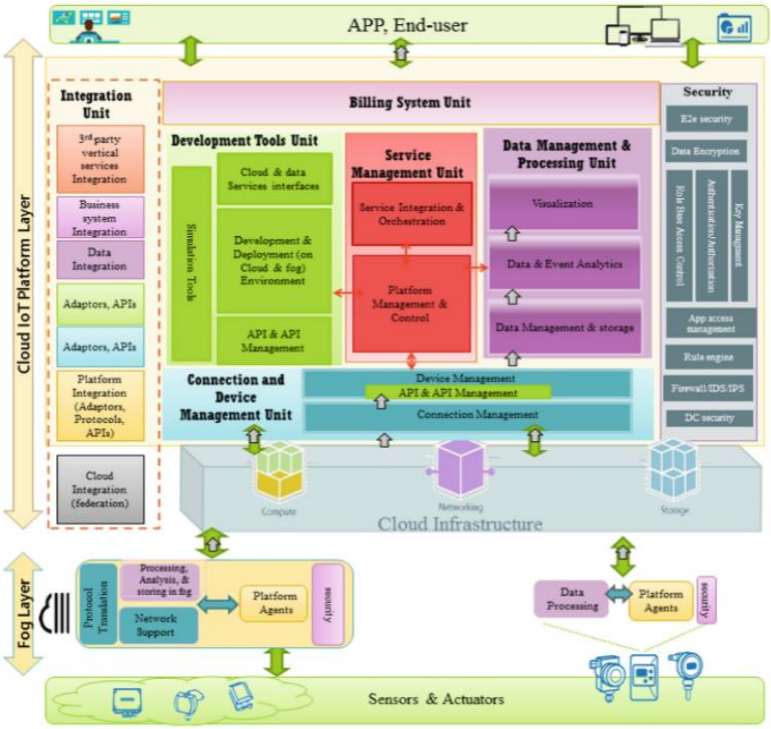

Figure 22: Fog-Enabled IoT platform Architecture by Asemani et al. (2019)

The role of iPaaS in this architecture for IoT could be on the whole Cloud IoT Platform layer (also referred to as the Platform on Cloud layer). The iPaaS will support the development, management, and processing of the integrations through only one platform. This will enable fog-enabled devices to integrate with other systems like 3rd party systems or business systems.

Conclusion

In the different architectures for IoT, a layered model can be seen. All the researched architectures consist of a Device layer. This layer involves the physical IoT devices, which can be either Sensors or Actuators. Within this layer, the IoT Gateway is also covered. This Gateway connects devices which cannot directly communicate with the Integration layer. In the Fog-Enabled IoT platform Architecture by Asemani et al. (2019), the Device layer further consists of the Fog layer.

The second layer is the Integration layer. This layer involves integrating the Device layer with the next layer based on events or topics. In the RA for IoT by Sobin (2020), this involves the Aggregation/Bus layer, combined with the Event Processing and Analytics component. In the IoT RA by Guth et al. (2016), this layer can be seen as the IoT Integration Middleware layer, connecting either with the Gateways or directly with the devices. In the Fog-Enabled IoT platform Architecture by Asemani et al. (2019), this layer can be seen as the Cloud IoT Platform Layer consisting of the Integration Unit, Development tools, Service & data management, and processing.

The third layer is the Presentation layer. This layer consists of the end-applications and user interfaces of the architecture. This layer is either accessible by users or other applications. In the RA for IoT by Sobin (2020), this layer can be seen as the combination of the Web Portal, Dashboard, and API management components. In the IoT Reference Architecture by Guth et al. (2016), this layer refers to the Application component. In the Fog-Enabled IoT platform Architecture by Asemani et al. (2019), this layer refers to the Application or End-user layer.

Additionally, the RA for IoT by Sobin (2020), and the Fog-Enabled IoT platform Architecture by Asemani et al. (2019) both involve a Security component. This security component is missing in IoT RA by Guth et al. (2016). To get a clear architecture of IoT, this Security component needs to be involved as well. A general IoT architecture can consists of three horizontal layers, and one vertical layer. An illustration of the layers of this IoT architecture is created and shown in figure 23.

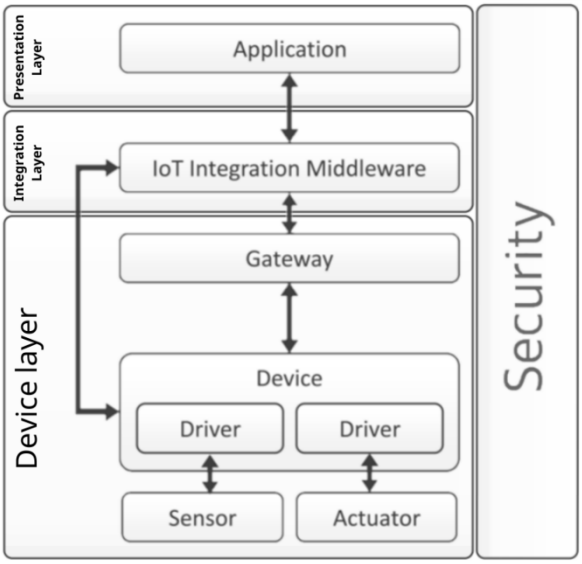

Figure 23: Illustration of a General IoT Architecture based on the IoT Reference Architecture by Guth et al. (2016)

A general overview of an IoT architecture, following the ArchiMate ADL, has been created and is given in figure 24. In this architecture, different components of the described architectures have been used. The core of this architecture is based on the IoT Reference Architecture by Guth et al. (2016). Further, the Web Portal, Dashboard, and API Management have been adopted from the RA for IoT by Sobin (2020). The core aspects of the Fog-Enabled IoT platform Architecture by Asemani et al. (2019) are already represented by the other architectures.

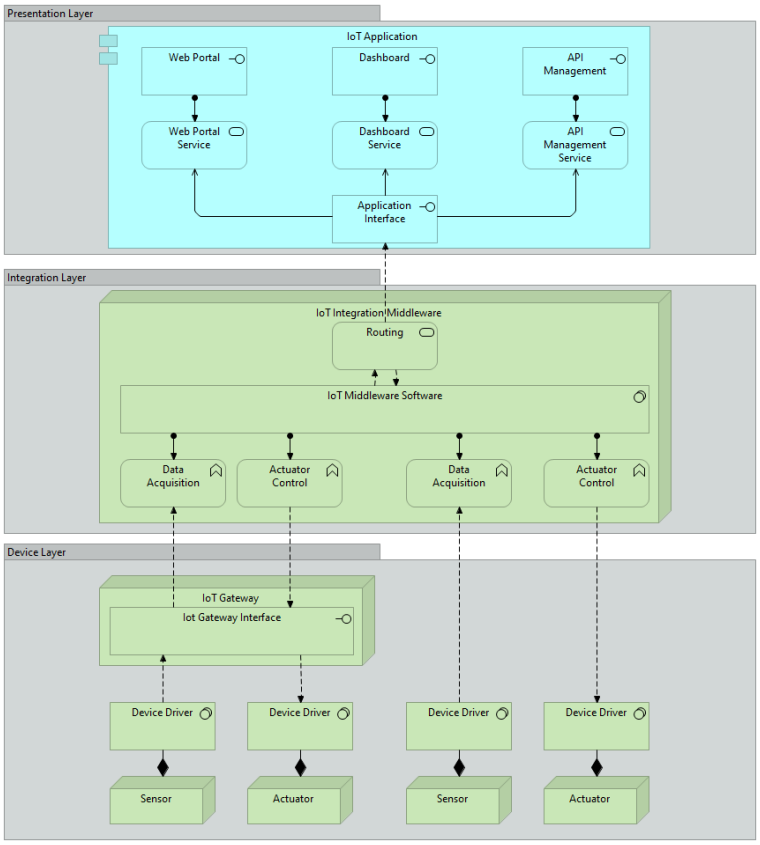

Figure 24: Simplified IoT architecture following the ArchiMate ADL

In the stated IoT architecture, the iPaaS architecture could be placed on the Integration Layer. The Device Layer could integrate with this Integration Layer with the help of a suitable Application protocol. These Application protocols are investigated in the following section.

5.1.2.3 Incorporation of Application Protocols

To finally connect the IoT to iPaaS, an investigation into how to integrate the found architectures, both IoT and iPaaS, needs to be conducted. As described by Sobin (2020), to integrate an IoT appliance, it needs to communicate with the help of different communication/application protocols. The term communication protocol refers to the protocol which enables devices to communicate directly with an IoT dependent gateway and fall out of scope of an iPaaS architecture.

This chapter will explore how to enable the incorporation of these application protocols in an iPaaS architecture. Hmissi & Ouni (2022) gave an overview of the five most used IoT application protocols. These protocols are MQTT, CoAP, DDS, AMQP, and XMPP. The given protocols will be further discussed below.

MQTT

The Message Queue Telemetry Transport (MQTT) protocol was introduced by IBM in 1999 and became a standard in 2013. Within MQTT, there are three components: the subscriber, the publisher, and the broker. The subscriber is a device that is interested in a specific topic, which is then informed by the broker when a publisher publishes to the topic of interest. The topics structure uses a forward's slash "/" as the delimiter between topics. An example in the case of IoT can be "home/kitchen/temperature". In this example "home", "kitchen", and "temperature" are all topics, all defining different levels (HiveMQ, 2019).

The difference between MQTT and AMQP is that the queue of AMQP stores the messages when it has not been retrieved by the receivers, while MQTT does not store these messages (Uy & Nam, 2019).

MQTT operates on the Transmission Control Protocol (TCP) transport layer, which provides reliable delivery of messages (Gorman, 2023). For further reliability of the messages, MQTT uses three Quality of Service (QoS) levels. The first QoS level, "0", involves "At most once" delivery, which means that the receiver does not confirm that the message have been received.

The second QoS level, "1", involves "At least once" delivery, which means that the receiver confirms to the publisher that the message have been received. The publisher stores and republishes the message if necessary.

The third QoS level, "2", involves "Exactly once" delivery, which means that the publisher guarantees that data is delivered exactly one time to the receiver. The difference between QoS 1 and 2 is the number of handshake actions. QoS 1 sends a confirmation if a message has been received totalling in two handshake actions, while QoS 2 ensures that there are no duplicate messages in a total of four handshake actions.

To get a better understanding of MQTT, a visual representation of the protocol has been created in the context of temperature and leakage sensors as publishers, and a client and a valve as subscribers. The created visual representation is given in figure 25.

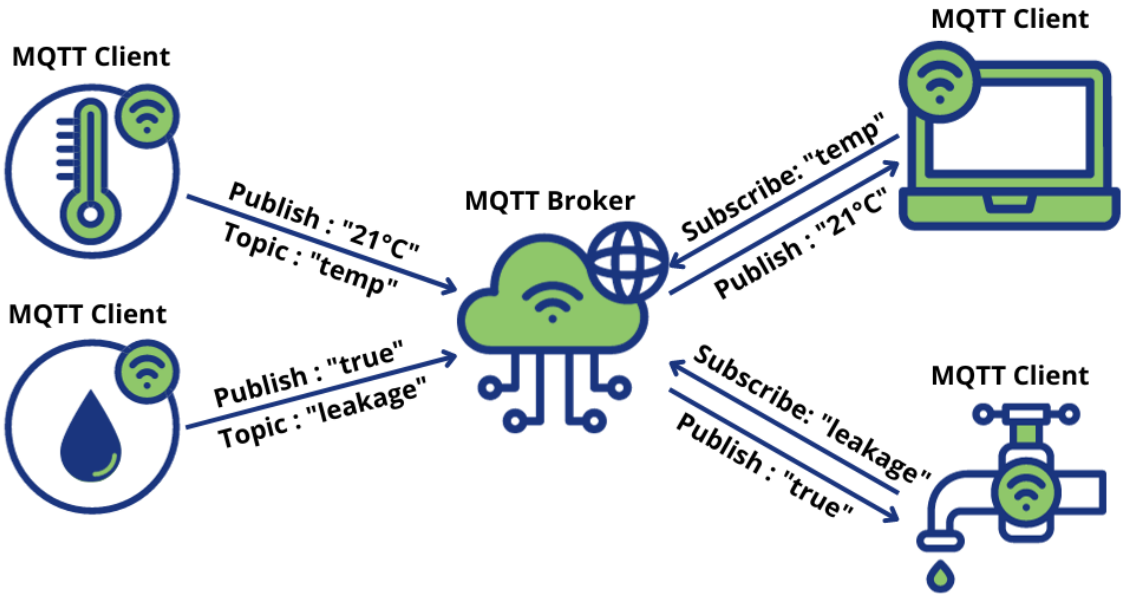

Figure 25: Overview of the MQTT Protocol with Temperature and Leakage Sensors

Di Paolo et al. (2023) provided an overview of different solutions for MQTT-brokers. The first solution being Mosquito. Mosquitto is an MQTT broker developed by Eclipse. Mosquitto is an Open-source MQTT broker/server, and fits the devices, sensors, and other IoT devices with low processing capacity. MQTT clients can connect to a Mosquitto broker to publish and subscribe to messages from specific topics (Dhall & Solanki, 2017). Uy & Nam (2019) provided an experiment for MQTT using the Eclipse Mosquitto MQTT Broker software.

The second solution is the EMQ X broker, which claims to be the leader in open source MQTT brokers for IoT because of its efficiency (Di Paolo et al., 2023). EMQ X is an open-source project from China,

started in 2013. There are three variants of EMQ X: the pure open-source broker, the Enterprise broker, and the private cloud solution (Koziolek et al., 2020).

The third solution is HiveMQ, which is widely used in automation and industrial systems (Di Paolo et al., 2023). HiveMQ was developed by the German dc-square company in 2012. In 2019 dc-square was renamed to HiveMQ and became an open-source broker. HiveMQ is provided in three editions: Community, Professional, and Enterprise. One of the appliances of HiveMQ is the connected car platform of BMW (Koziolek et al., 2020).

The fourth solution is Moquette, which is a very light open-source broker. This broker is less-known, and less-used, than the other stated brokers (Di Paolo et al., 2023).

The fifth solution is Aedes. Aedes is the successor of MoscaJS, which is a powerful MQTT broken written in Node. Bhatt et al. (2022) provided the details of the implementation of the MQTT for IoT in Advanced Metering Infrastructure Network of Smart Grid Systems. For this implementation, they made use of this Aedes broker.

AMQP

The Advanced Message Queuing Protocol (AMQP) was developed by OASIS. This protocol is reliable and enables communication via message delivery. The AMQP protocol is comparable to the MQTT protocol, both following the Publish/subscribe pattern, and both using the same three levels of Quality of Service. AMQP, just like MQTT, also operates on the TCP transport layer.

AMQP enables publishers to send messages to an Exchange. This Exchange routes the messages to different queues in the system, based on routing keys, binding keys, or Exchange types. The main difference between MQTT and AMQP is that in AMQP, messages will be stored in the queue until it is consumed (Uy & Nam, 2019).

To get a better understanding of AMQP, a visual representation of the protocol has been created with one publisher and two consumers. The created visual representation is given in figure 26.

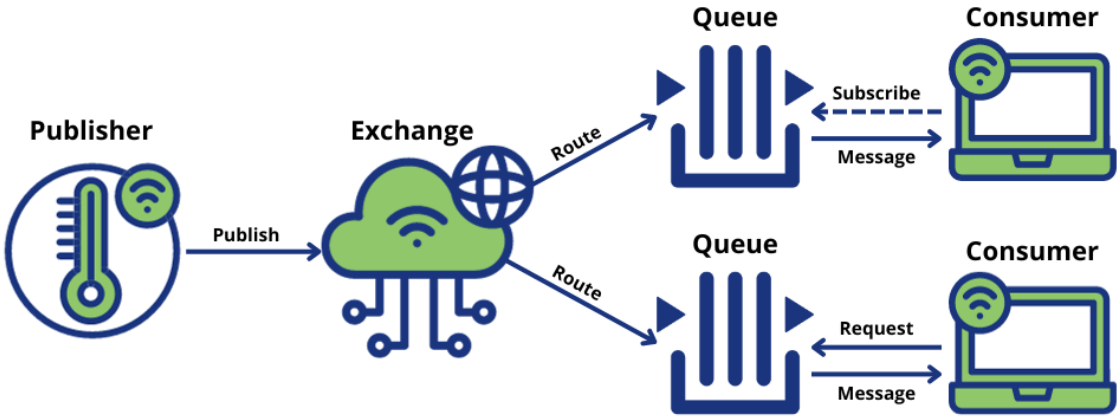

Figure 26: Overview of the AMQP Protocol

John & Liu (2017) mentioned different AMQP solutions. The first solution being Apache ActiveMQ. This solution incorporates the Java Message Service (JMS) specification, which is an enterprise messages standard included in the Java Enterprise edition. In addition to AMQP, ActiveMQ also supports other protocols like MQTT, Stomp, TCP, UDP, and XMPP. An ActiveMQ broker can be set up as a federated cluster. This federation enables exchanges or queues on one broker to receive the published messages of an exchange or queue on another broker (John & Liu, 2017).

The second solution for AMQP is ZeroMQ, which is a broker-less, Smart-endpoint, Dumb network philosophy (John & Liu, 2017). ZeroMQ is an open-source broker developed by iMatix, and the evolution and maintenance are community driven (Lauener & Sliwinsky, 2017).

The third solution for AMQP is RabbitMQ, which was developed by VM-Ware. RabbitMQ is a popular AMQP broker, which is easy to use and deploy. Just like ActiveMQ, this RabbitMQ uses a federation of brokers to achieve capacity scaling (John & Liu, 2017).

The fourth solution for AMQP is Apache Qpid. Qpid provides two different versions for message brokers: A Java version, and a C++ version. Just like ActiveMQ and RabbitMQ, Qpid uses a federation of brokers to achieve capacity scaling (John & Liu, 2017).

DDS

The Data Distribution Service (DDS) was developed by the Object Management Group (OMG). DDS also follows a Publish/subscribe pattern for real-time machine to machine (M2M) communications and can operate on both the TCP and UDP transport layers. DDS does not need a broker, instead it uses multicasting to achieve the high reliability. This broker-less architecture can be seen as a fitting solution for the real-time connection of IoT. Instead of a broker, DDS uses Data Writers (DW) and Data Readers (DR). DDS keeps the consumers of the data anonymous since publishers do not get to know who consumes the data. Further, DDS is a reliable protocol because it does not have a Single Point of Failure. DDS is not suitable for constrained IoT devices, because it was invented for Industrial IoT applications.

To get a better understanding of DDS, a visual representation of the protocol has been created with two data writers, and two data readers. The created visual representation is given in figure 27.

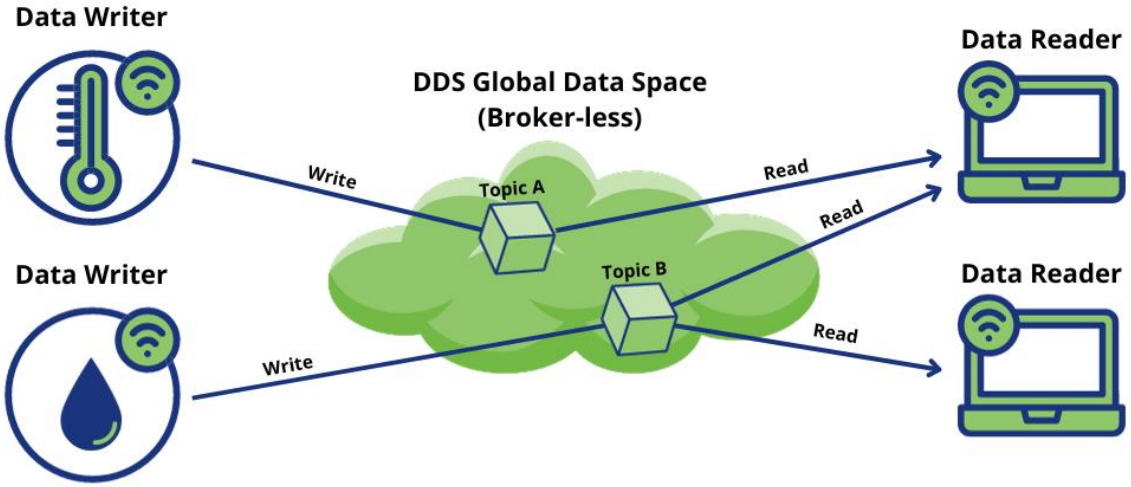

Figure 27: Overview of the DDS Protocol

Maggi et al. (2022) provided an overview of different solutions for DDS Networks. The first solution being Fast DDS, which was developed by eProsima in 2014. Fast DDS is a C++ based open-source implementation of DDS middleware. Fast-DDS supports several transport protocols like UDP (v4 and v6), TCP (v4 and v6), and Shared Memory (SHM). Further, Fast DDS is faster than a Cyclone DDS implementation (Stevanato et al., 2023). Fast DDS has different types of support. The Dynamic support type decides at runtime how to serialize the message types while the Static support type shifts the computation to compile time (Betz et al., 2023).

The second solution is Cyclone DDS. This solution was developed by the ADLINK-driven Eclipse Foundation project in 2011. Which is described as a mature and fully featured implementation of DDS hosted at the Eclipse Foundation, the same foundation hosting the Mosquitto MQTT broker as mentioned before (Desbiens, 2023). Cyclone DDS is widely adopted in industries like robotics and autonomous vehicles in car specific projects (Liang et al., 2023).

The third solution is Open DDS. This solution was developed by OCI in 2005, and is a C++ based open-source DDS implementation. Open DDS provides the Hot-swap function which allows users to switch the publishers of a specific topic in real-time to avoid parts of the grid getting unavailable. Further, Open DDS provides the Extensible Transport Framework (ETF), which enables TCP, UDP, multicast, SHM, and Real Time Publish Subscribe (RTPS) UDP (Krüger et al., 2018).

The fourth solution is Connext DDS. This solution was developed by Real Time Innovation (RTI) in 2005. RTI Connext DDS is known for its validated real-time performance, robustness, and extensive supporting tools. According to Brückner & Swynnerton (2014), the key value of RTI Connext DDS is its business model. This business model involves an Open Architecture aiming to expand the selection of applications, reduce procurement cost, integrate systems, and more. Audi is one of the consumers of RTI Connext DDS, using it for an Automotive Hardware-in-the-Loop (HiL) test platform (Brückner & Swynnerton, 2014).

The fifth solution is CoreDX DDS. This solution was developed by the Twin Oaks Computing Incorporation in 2009. CoreDX DSS can run on enterprise scale servers, but it does not require costly computing platforms. With CoreDX DDS, a DDS application makes it possible to be run on standard Intel-based desktops and laptops (Shalchian & Ravanmehr, 2016).

XMPP

The Extensible Messaging and Presence Protocol (XMPP) was developed by the Jabber open-source community. XMPP enables the users to communicate by sending instant messages over the Internet, operating on the TCP transport layer. XMPP is secure and connects a server and its clients with the help of XML stanzas. These XML stanzas are pieces of code divided in three components: presence, message, and iq (info/query).

Hornsby et al. (2009) provided a detailed overview of these stanzas. The “<presence/>” stanza follows the basic publish/subscribe pattern. These stanzas are used for the information about the network availability of entities. The “<message/>” stanza follows a push mechanism where an entity sends asynchronous information to other entities. This stanza is optimized for real-time delivery, but also supports storing and late delivery. The <iq/> stanza follows a mechanism where entities make requests and receive responses from each other.

XMPP, just like some other protocols, supports a Publish/subscribe pattern. Further, XMPP also support a Request/response pattern. Within IoT, it does not matter which operating system is used for XMPP (Hmissi & Ouni, 2022).

To get a better understanding of XMPP, a visual representation of the protocol has been created with a total of four clients, connected through two XMPP servers. The created visual representation is given in figure 28.

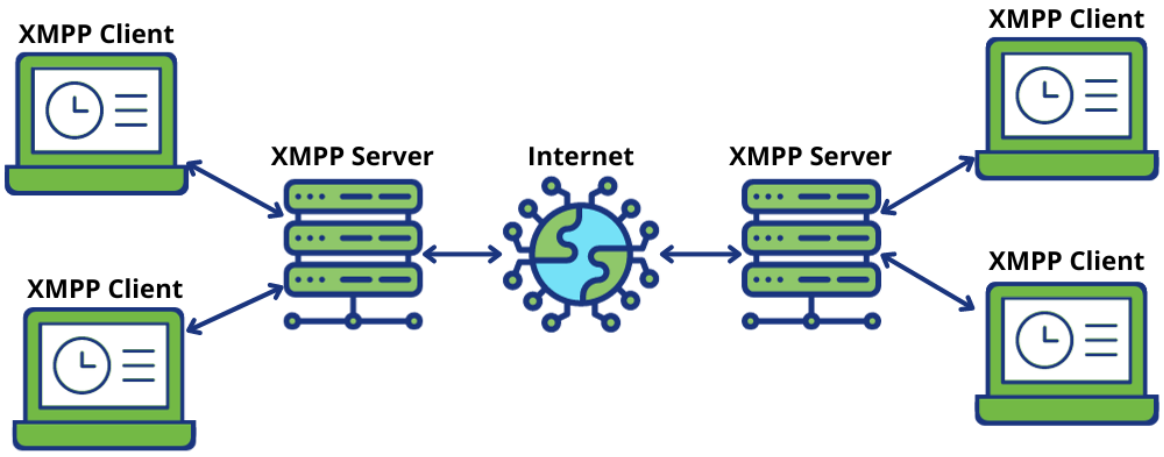

Figure 28: Overview of the XMPP Protocol

The XMPP organization mentioned different solutions for the XMPP server (XMPP Organization, sd). The first solution is eJabberd, which can be seen as one of the major XMPP server software solutions according to Celesti et al. (2017). Further, eJabberd supports a robust, scalable, and extensible real-time platform providing an XMPP Server, MQTT Brokers, and Session Initiation Protocol (SIP) services (eJabberd, sd).

The second solution is Isode M-Link. The key features of M-Link are security, reliability, and special functionality for large public deployments. Isode provides software for solutions in the government, military, and aviation authority over 150 countries. The M-link servers does not support standards for operations over constrained networks (Isode, sd).

The third solution is OpenFire, which is an open-source, Java-based, publish/subscribe XMPP server. OpenFire uses the Jabber Identifier (JID) for referencing a device on a gateway (Viswanath et al., 2016). Aros & Torres (2018) mentioned that OpenFire is a free, big community supported XMPP server. OpenFire was designed to be a centralized system for serving many clients (Zhang et al., 2016).

The fourth solution is the Tigase XMPP Server. Tigase is a popular Java-based XMPP server with active development and a great community (Watkin & Koelle, 2016). This solution can be used to build Instant Communication (IC) systems. Tigase in its most basic form can be seen as a chat server, but Tigase can do much more than just that (Tigase.net, sd).

CoAP

The Constrained Application Protocol (CoAP) was developed by the Internet Engineering Task Force (IETF). CoAP is based on the Representational State Transfer (REST) protocol. REST facilitates a simple way to exchange data between the clients and servers over the Hypertext Transfer Protocol (HTTP), which is the foundation of the World Wide Web as it is known now. CoAP modifies some of the functionalities of HTTP in a way that it meets the requirements of IoT, such as low power consumption and operation in the presence of lossy and noisy links. CoAP enables small devices with limited capabilities on power, computation, and communication to utilize RESTful interactions. CoAP consists of two layers: the messaging layer, and the request/response layer. CoAP operates on the UDP transport layer.

CoAP main usages is in constrained environments, with constrained devices, and constrained networks. The request/response messages use an URI, with GET, POST, PUT, and DELETE actions to keep it lightweight. Further, CoAP has two modes for messaging. The first mode, Piggybank Mode,

involves that the server sends it response directly after receiving a request. The second mode, Separate Mode, involves that the server indirectly communicates between the clients and server. The response in Separate Mode is a separate message, and it may take time for the server to send this message (Hmissi & Ouni, 2022).

To get a better understanding of CoAP, a visual representation of the protocol has been created with different clients connected in a constrained environment through the internet. The created visual representation is given in figure 29.

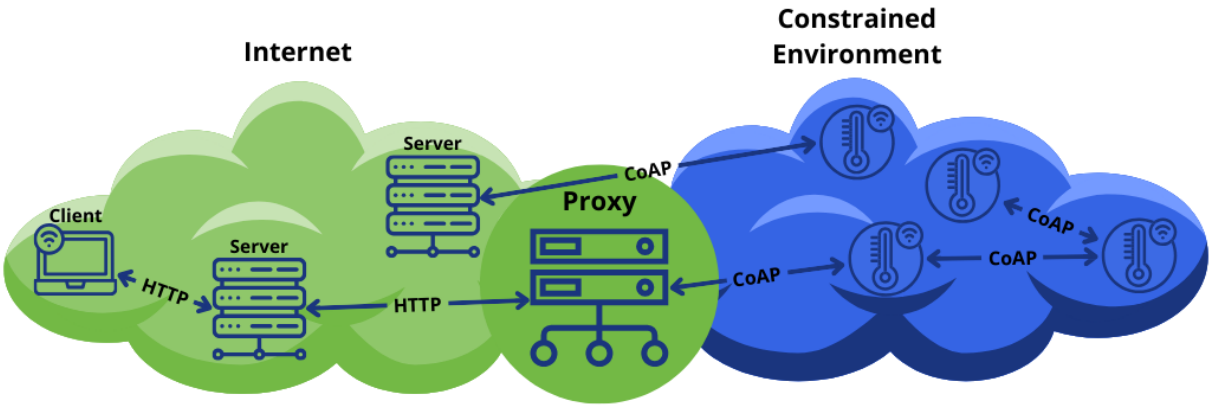

Figure 29: Overview of the CoAP Protocol

Iglesias-Urkia et al. (2019) provided an overview of different CoAP implementations. The first one being libCoAP, which is a library designed to fit a wide range of devices. libCoAP supports several extensions, and the source code involves complete examples of the implementation. libCoAP incorporates an interface, simplifying the process of adding new resources. In this library, the response codes are managed by itself, which means the developers only need to involve the resource name, request type, and a link to the request handler.

The second implementation is the Simple Media Control Protocol (SMCP), which is a C-based stack used in an embedded environment (Bernstein et al., 2021). The aim of SMCP is to be implemented in a wide range of devices, from simple sensors to Linux based devices. SMCP provides clients with a command line called SMCPTL. Like libCoAP, only the handlers need to be created and the resources need to be added with the help of an interface (Iglesias-Urkia et al., 2019).

The third implementation is MicroCoAP, which is a C-based limited library, with its aims on microcontrollers. This implementation only supports server sides, having limited features. MicroCoAP does not support the DELETE action, only the POST, PUT, and GET. Adding new resources to the server must be done by defining and adding a new resource to the resource array, along with its handlers, the rest is managed by the library itself (Iglesias-Urkia et al., 2019).

The fourth implementation is FreeCoAP. This is also a C-based CoAP library targeting Linux devices. This implementation does not include an interface for the resource handling, thus the process of adding new resources, and the handling of the responses can be tricky (Iglesias-Urkia et al., 2019)

The fifth implementation is Californium created by the Eclipse Foundation, the same foundation hosting the Mosquitto MQTT broker, and the Cyclone DDS as mentioned before. Californium is described as a complete Java library for devices that are not too constrained. Iglesias-Urkia et al. added that the implementation is very mature, making it easy to add and manage new resources through an interface, while the library handles the rest (Iglesias-Urkia et al., 2019).

The sixth implementation is Node-CoAP, which is a JavaScript based library for targeting the Node.js platform. Node-CoAP provides clients and server sides, following a stable version of the Request For Comment (RFC). The developer is responsible for the handling of the resources and response codes, without an interface. This means that the handling of resources and requests needs to be made by the application, not by the library (Iglesias-Urkia et al., 2019).

The seventh implementation is CoAPthon, which is a Python based library. CoAPthon offers support on both the client and server side. A new Python class needs to be created for each new resource. The library is managed by itself, resulting in ease of use for the developers (Iglesias-Urkia et al., 2019).

The eighth implementation is CoAPy, which is also a Python based library. CoAPy follows an old concept of CoAP, which makes it incompatible with other libraries. A new Python class needs to be created for each new resource, and the application is responsible for handling the response codes (Iglesias-Urkia et al., 2019).

Integrating IoT on iPaaS

To integrate IoT appliances on an iPaaS architecture, an Event Streaming integration pattern is needed. This pattern is used by most of the described IoT architectures above. A popular solution for incorporating Event Streaming on an iPaaS architecture is using Apache Kafka (Chatt, 2019). Kafka is a scalable, fast, reliable, and durable solution for Event Streaming which uses Topics, Producers, Consumers, Connectors, Stream Processors, Brokers, and a Zookeeper (Hiraman et al., 2018). Figure 30 shows the general architecture framework of Apache Kafka.

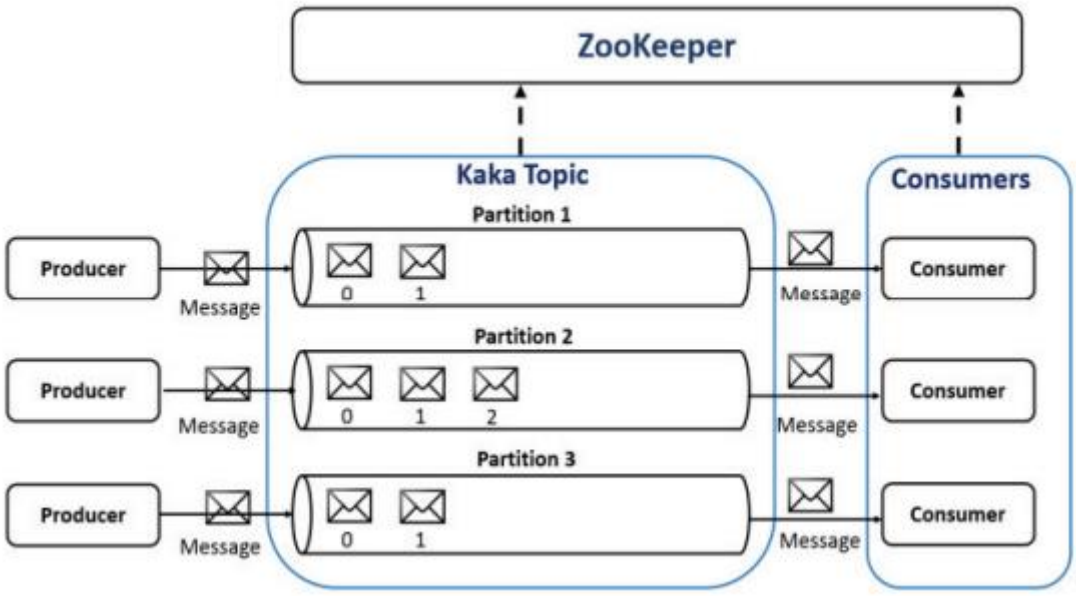

Figure 30: Apache Kafka Framework by Hiraman et al. (2018)

The first component in this framework is the topic. This topic is a system which stores and publishes the messages on the Kafka Broker. The producers in the framework publish messages to a specific Kafka topic. The topics structure uses a forward's slash "/" as the delimiter between topics.

The second component is this framework is the producer. These producers publish messages to one or more specific Kafka topics on the Kafka broker.

The third component is the consumer. These consumers read the messages from the Kafka broker. This is done by subscribing to one or more specific Kafka topics and pulling the data from these topics.

The fourth component is the connector. This connector is responsible for pulling the data from the producers and delivering the data to the consumers and the stream processors.

The fifth component is the stream processor. This stream processor is an application which can transform the messages of a topic. The stream processer next streams the data to another Kafka topic within the same Kafka cluster.

The sixth component is the Kafka Broker. This broker is stateless, which means that it consumes messages without maintaining any information about past messages. A so-called Zookeeper is used for maintaining the state of the Kafka clusters.

The final component is the Zookeeper. This Zookeeper manages and coordinates the Kafka Broker. It is also used to notify the producers and consumers about the Kafka broker or about a failure within the Kafka broker.

Connecting the IoT Protocol to the Event Streaming protocol

To finally integrate different IoT appliances into the above-mentioned Event Streaming pattern, the MQTT protocol could be used. It is chosen to use MQTT because as of today, MQTT seems to be leading the way in IoT applications (Priyadarshi & Behura, 2018). Farhan et al. (2022) further mention that MQTT is one of the most used protocols within the field of IoT. To connect an MQTT client to Apache Kafka, two options have been identified by Hettig (2021). The first option is using an MQTT Broker which connects to the Kafka Broker with the help of the MQTT Broker source connector.

The second option is the use of an MQTT Proxy. This option connects the MQTT Clients to the Kafka Broker without the use of an MQTT Broker. The MQTT Proxy forwards the incoming payload to a pre-defined Kafka Topic. This solution is a more cost-effective option, because not the whole MQTT-broker is needed.

A general architecture for enabling IoT integrations on an iPaaS architectures was created and is shown in figure 31. This architecture follows the concept of the MQTT Proxy for an iPaaS architecture using the Kafka Broker solution for the Event Streaming.

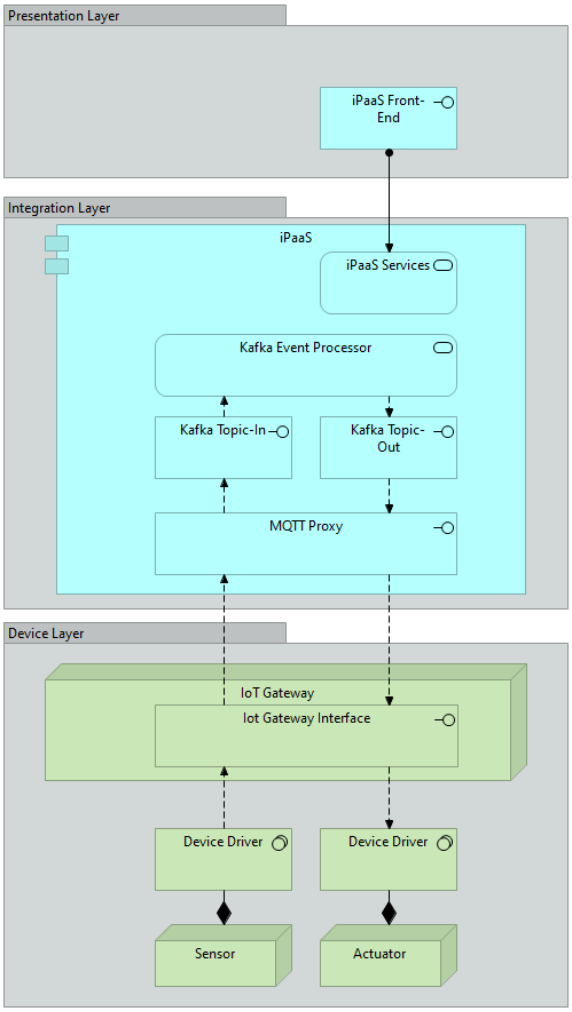

Figure 31: Combined IoT and iPaaS Architecture

5.1.3 Semantic Interoperability in iPaaS architectures

The second research question addressed in this study is “How can Semantic Interoperability be incorporated in iPaaS architectures?”. To better understand Semantic Interoperability in iPaaS architectures, the ArchiMate Architecture Description Language is used. The motivation and description of this method is given in the methodology section of this chapter.

Semantic Interoperability in the context of computer science can be stated as “the ability of computer systems to exchange data with unambiguous, shared meaning” (Network-Centric Operations Industry Consortium - NCOIC, 2008).

Different approaches for achieving Semantic Interoperability have been identified by Vermesan (2018), as seen in figure 32. These approaches have been classified by three types of Semantic Interoperability. The first type is By chance, which involves that each platform is free to use the models they like. These platforms are only interoperable with the other platforms that, by chance, use the same model. The second type is By standardization, which involves that there is a standardization for parts of the used models. The last type is By mapping, which involves that mapping logic is used to make a translation between different models.

The most common approach in achieving Semantic Interoperability is to use an ontology, which follows the idea of the Semantic Web by Berners-Lee et al. (2001). The core of the Semantic Web technologies is the Resource Description Framework (RDF), a lightweight meta data model for ontology specification, and the SPARQL Protocol and RDF Query Language (SPARQL, which is a recursive acronym). All these technologies are standardized by the World Wide Web Consortium (W3C).

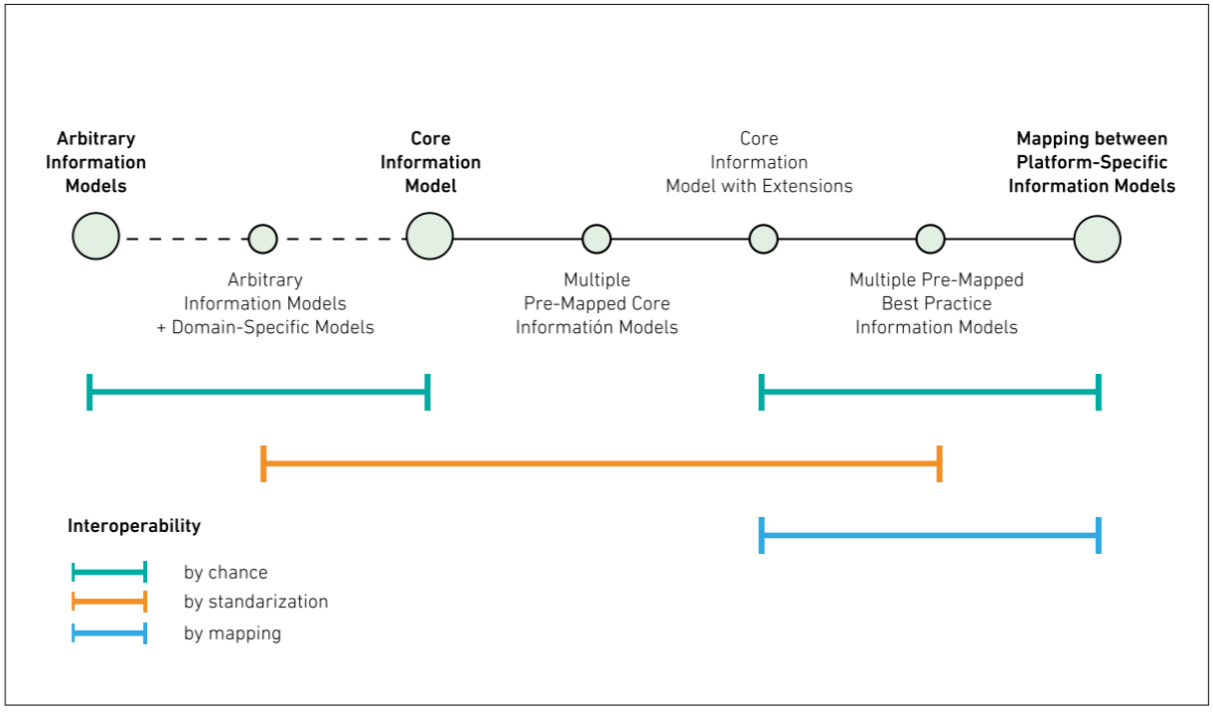

Figure 32: Approaches for Semantic Interoperability by Vermesan (2018)

Arbitrary Information Models (+ Domain-Specific Models)

The first two approaches mentioned by Vermesan (2018) are the Arbitrary Information Models (AIM), and the AIM + Domain-Specific Models. They both do not solve the problem of Semantic Interoperability, but they provide the mechanisms needed to address Semantic Interoperability.

Therefore, these approaches allow a foundation for solutions on a different level. Both approaches can be seen as interoperability by chance, meaning that they are only interoperable with other platforms that, by chance, use the same model. The AIM + Domain-Specific Models can also be classified as Interoperability by standardization, meaning that there are standardizations for parts of the used models.

Core Information Model

The third approach mentioned by Vermesan (2018) is the Single Core Information Model (CIM). This solution is most used among existing platforms. This CIM should be used by all the platforms in an ecosystem, meaning that the platforms can only expose data which fits into the CIM. Custom extensions are prohibited, meaning that this custom data cannot be interoperable with other platforms. This approach can be seen as interoperability by chance, meaning that they are only interoperable with other platforms that, by chance, use the same model. Further, this approach can be seen as interoperability by standardization, meaning that there are standardizations for parts of the used models. An illustration was created to better explain this concept, as seen in figure 33.

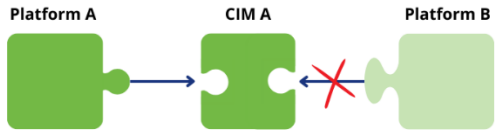

Figure 33: Illustration of the Core Information Model

The advantage of this solution is that it easy to use and implement, because only one information model needs to be considered. The downside is that it can be difficult defining a single information model conforming to all the platforms. Further, these single information models could become extensively complex, compromising all exchanged data in an ecosystem. The CIM would also exclude platforms not following the CIM, with no scalability for integrating future platform.

Multiple Pre-Mapped Core Information Models

The fourth approach mentioned by Vermesan (2018) is the Multiple Pre-Mapped CIMs, making it easier for platform owners integrating internal information models supporting not only one single CIM, but multiple CIMs. Many existing platforms could participate, following well-established CIMs. These well-established CIMs should be ontologies which are widely used, to make these ecosystems more interoperable. To enable interoperability between the platforms in the ecosystem, the CIMs need to be mapped onto each other beforehand. This approach can be seen as interoperability by standardization, meaning that there are standardizations for parts of the used models. An illustration was created to better explain this concept, as seen in figure 34.

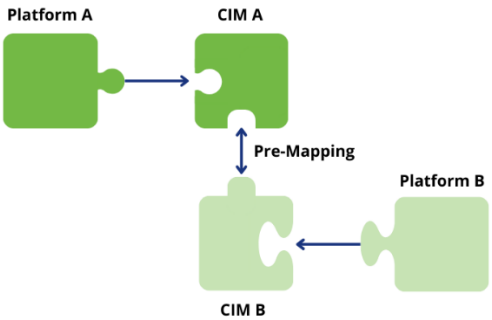

Figure 34: Illustration of Multiple Pre-Mapped Core Information Models

The advantage of this approach is the flexibility, because future CIMs and mappings can be added over time. Further it does not make use of a single CIM, excluding less platforms from the ecosystem. The downside could be that still some platforms are excluded which do not match the pre-mapped CIMs.

Core Information Model with Extensions

The fifth approach mentioned by Vermesan (2018) involves a CIM with Extensions. This approach focuses on creating an information model that is both highly abstract and detailed, serving as a foundation for various platform-specific representations. This approach defines high-level classes and their relationships, serving as points for extensions unique to each platform. These extensions can either directly utilize the provided classes or define subclasses with platform-specific properties. In addition to high-level classes, the core model may include general properties like ID or name. This results in a minimalistic core that all platforms must follow, with extension points for custom requirements. Platforms with different extensions can understand each other through the core model, but they must establish semantic mappings for custom extensions. This approach can be seen as interoperability by standardization, meaning that there are standardizations for parts of the used models. Further, this approach can be seen as interoperability by mapping, meaning that mapping logic is used to make a translation between different models. Finally, this approach can also be seen as interoperability by chance, meaning that they are interoperable with other platforms that, by chance, use the same model. An illustration was created to better explain this concept, as seen in figure 35.

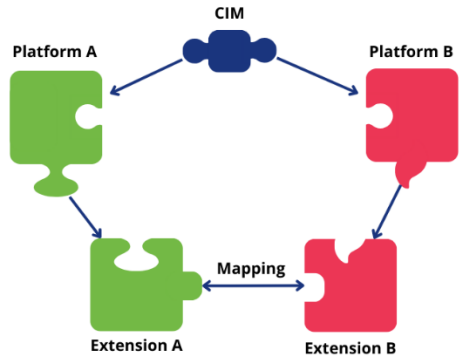

Figure 35: Illustration of the Core Information Model with Extensions

The advantages of this approach are that it provides basic interoperability through the minimalistic core model, offers flexibility through custom extensions, and is generally well-received by adopters for combining out-of-the-box interoperability with support for complex scenarios. The downsides are the needs for semantic mapping when different platforms want to understand custom extensions. Further, the complexity of defining such mappings, and the complexity of designing the core information model can also be seen as downsides.

Multiple Pre-Mapped Best Practice Information Models

The sixth approach mentioned by Vermesan (2018) involves using Pre-Mapped Best Practice Information Models, a modification of the Multiple CIMs concept. In this approach, the provided information models are considered best practice information models rather than CIMs. Platforms are not obliged to follow to these models, giving the platforms freedom to choose their own. However, if a platform decides to use one of these best practice information models, it gains instant interoperability with other platforms aligned with the same model. This approach can be seen as interoperability by standardization, meaning that there are standardizations for parts of the used

models. Further, this approach can be seen as interoperability by mapping, meaning that mapping logic is used to make a translation between different models. Finally, this approach can also be seen as interoperability by chance, meaning that they are interoperable with other platforms that, by chance, use the same model. An illustration was created to better explain this concept, as seen in figure 36.

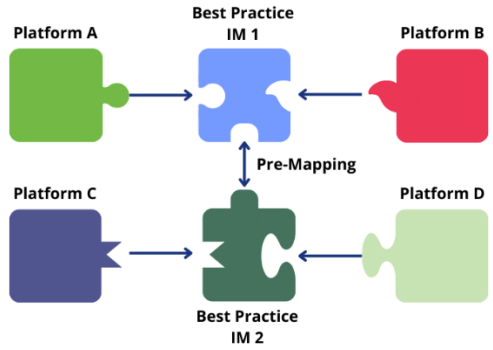

Figure 36: Illustration of Multiple Pre-Mapped Best Practice Information Models

The advantages of this approach include that there are no limitations on information models, making it more usable for platform owners, enabling broader interoperability through aligned best practice information models. On the downside, there is a lack of initial interoperability between platforms when pre-mapped information models are not used. Creating semantic mappings for interoperability can also be a complex task, requiring additional effort from developers and platform owners.

Mapping between Platform-Specific Information Models

The seventh approach mentioned by Vermesan (2018) involves mapping between platform-specific information models, with no central CIM in place. Each platform independently provides its own information model, and interoperability is achieved through mappings between these platform-specific models. This approach can be seen as interoperability by mapping, meaning that mapping logic is used to make a translation between different models. Finally, this approach can also be seen as interoperability by chance, meaning that they are interoperable with other platforms that, by chance, use the same model. An illustration was created to better explain this concept, as seen in figure 37.

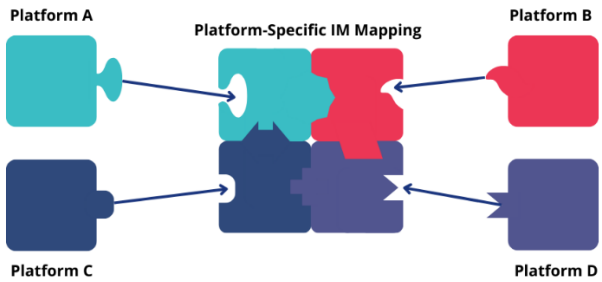

Figure 37: Illustration of Mapping between Platform-Specific Information Models

The advantages of this approach are that it supports all possible information models and platforms, with the flexibility to add mappings in the future to increase interoperability. The downsides can be that it lacks initial interoperability until the mappings are defined. Creating semantic mappings can be complex, requiring additional effort from developers and platform owners. Moreover, the system does not possess an inherent understanding of the data it processes.

Semantic Adapter by Zehnder et al. (2020)
One solution for bridging the semantic gap is the implementation of a Semantic Adapter. Zehnder et al. (2020) suggested an adapter model for bridging the gap in the context of Industrial IoT. These adapter models enable the description of time series data sources. The descriptions of the adapter are provided in Resource Description Framework (RDF) format, serialized as JSON-LD (JSON for Linked Data). This adapter model can be seen in figure 38.

The Adapter is the core of the model, which has a Stream Grounding for describing the protocol and format which is used to publish the harmonized data. If unified data is needed to be sent to a message broker, the Adapter could make use of Transformation Rules. The adapter support both Data Sets, and Data Streams, noted as {Stream, Set} in the model of Zehnder et al. (2020).

Within the Data Steam Adapters, two types of adapters can be distinguished. The first one is the Generic Data Stream Adapter, which is a combination of the Data Stream Protocol for connecting to different data sources and formats required for converting the data into the internal representation. The other adapter type is the Specific Data Stream Adapter. This adapter is used for the data sources not compliant with these standards. These Specific Adapters could also be used for custom solutions and implementations of other data sources.

The user configurations of an adapter are provided by the Static Properties. These could be used for the configurations of the formats, protocols, and adapters. The Configurations of the Adapters are stored in the Adapter Templates. This could be something like the required API keys.

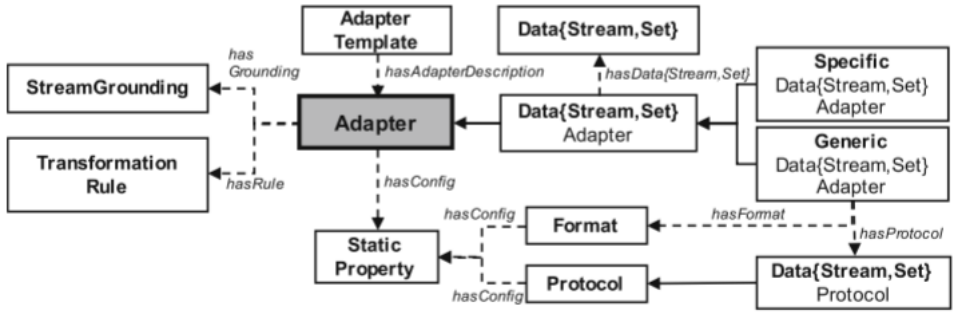

Figure 38: Adapter Model by Zehnder et al. (2020)

The Transformation Rules could be further distinguished into another model, as seen in figure 39. these rules make it possible to transform, reduce, or anonymise the data on the adapter. The first transformation rule is the Stream Transformation Rule for aggregating data and removing duplicates. The second transformation rule is the Schema Transformation Rule for changing the schema of the data. The last transformation rule is the Value Transformation Rule for changing the values of the data.

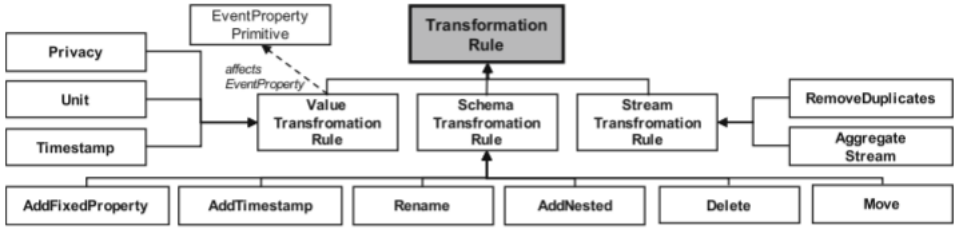

Figure 39: Transformation Rule of the Adapter Model by Zehnder et al. (2020)

Further, Valderas & Torres (2023) described the Semantic Adapter in the context of IoT-Enhanced business processes. The communication between microservices was done with the help of semantic messages published in an asynchronous event bus. Each microservice needed an adapter for transforming the messages into the native data structure of each system. In the case of IoT microservices, these Semantic Adapter transforms the raw captured data into the desired format. This adapter was implemented using Java Libraries.

Singh et al. (2021) further stated the definition of a Semantic Adapter as "a data ingestion tool that provides semantic abstraction, annotation and raw data transformation, coming from the IoT device layer into the data layer in an automated way, based on predefined and learned ontology models."

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as a Core Information Model with Extensions. The Core Information Model in this case is the internal representation of the chosen ontology, the extensions are the data sources which are not compliant with the chosen standards. The data is mapped from the source Information Model to the Core Information Model.

MASSIF-platform by Bonte et al. (2017)
Bonte et al. (2017) provided the Modular, Service, Semantic & Flexible (MASSIF) platform. This platform was designed for enrichment and reasoning of IoT data with the help of ontologies to semantically annotate raw data. The architecture of MASSIF can be seen in figure 40. This architecture consists of four layers. The Input layer receives the low-level data in JSON format from the devices on the Gateway. This data is moved to the Matching Service which decides which Context Adapter is selected for semantically annotating the low-level data based on the tag in the from the Gateway received JSON message.

The Semantic Annotation layer consists of the Context Adapters, which receives low-level raw data from the Matching Service. This layer next semantically annotates the data. There can be multiple Context Adapters activated to annotate the different kinds of data, with each Context Adapter annotating a specific kind of received data. The data will be converted to OWL individuals, after which the data will be pushed to the Semantic Communication Bus (SCB). The Context Adapters can also be virtual, which means that they do not receive data from the Matching Service, but they annotate data captured from existing sources like Really Simple Syndication (RSS) feeds.

The Flow Decision layer is made up by the SCB, which provides a publish-subscribe pattern based on high-level ontology concepts. The Context Adapters from the Semantic Annotation layer can be seen as the publishers, while the Services on the Service layer can be seen as the subscribers. These publishers publish their data in the form of OWL Axioms, which are statements that are asserted to be true in the domain which is being described (W3C, sd). When data gets published, it gets added to the ontological model of the SCB. With the help of semantic reasoning on the ontology, the published event type is retrieved. This matches the filter rules of the subscribed services. After the data is forwarded, it gets removed from the ontological model.

The Service layer consists of the different services within the ecosystem. These services subscribe to the SCB with the help of one or more filter rules. After the consumed data is processed, inferred knowledge is published to the SCB. This is to notify other services about the insights. Each of these services contain their own ontology and reasoner. With the filter rules, the data each service receives is limited, which can conclude in slower reasoning with an increasing dataset size. A special Notification Service gathers all the knowledge from other services,

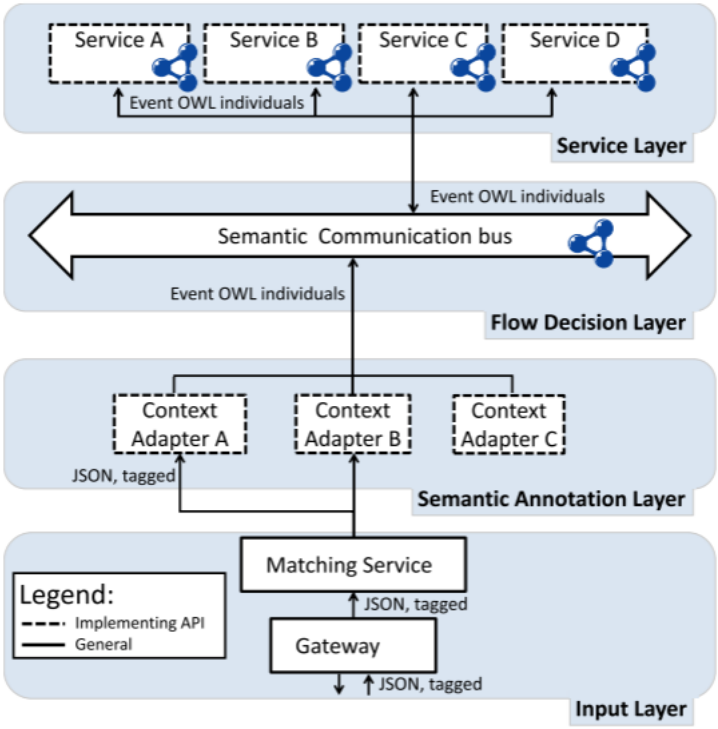

Figure 40: Architecture of the MASSIF Platform by Bonte et al. (2017)

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as the Mapping between Platform-Specific Information Models. There is no Core Information Model in this case, and the mapping is done by the different context adapters.

IoT Platform Semantic Mediator (IPSM) by Ganzha et al. (2018)
Ganzha et al. (2018) provided an IoT Platform Semantic Mediator (IPSM) for achieving Semantic Interoperability between IoT platforms. Ganzha et al. gave the term Translator which is a software artifact for accepting data expressed in one semantics, and produce data expressed in different semantics, while keeping the same meaning.

Within this IPSM, three solutions have been considered. The first approach involves One-to-One Alignment Translators between ontologies. This approach involves establishing the alignment between two platforms as seen in figure 41. Ganzha et al. used the example of four different IoT platforms (P, R, S, and T), using different semantics. Each of these platforms has its own ontology (OP, OR, OS, and OT). Between these platforms, the ontologies are translated. For example, from platform P to platform S, the ontology of OP will need to be translated into ontology OS. The formula for calculating the amount of Alignment Translators is $t = \frac{n\,(n-1)}{2}$, where n is the number of platforms in an ecosystem, and t is the total number of One-to-One Alignment Translators in an ecosystem. In the example of four platforms, this will include six One-to-One Alignment Translators.

According to Ganzha et al. (2018), the advantage of these One-to-One alignments is the good quality of the ontology translations, because these alignments are directly generated between two systems. The disadvantage of this approach is the scalability. Considering the stated formula, an ecosystem with fifteen platforms would need to create fifteen new One-to-One Alignment Translators when a new platform is added. This makes the total amount of One-to-One Alignment Translators 120 instead of the previous 105.

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as the Multiple Pre-Mapped Core Information Models. The Core Information Models in this case are the platform specific Information Models. The data of the Platform Specific Information Model is mapped to the target Core Information Model.

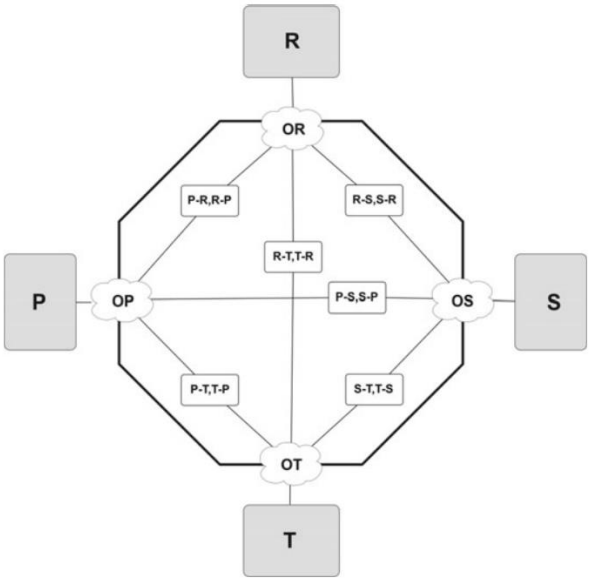

Figure 41: One-to-One Alignment by Ganzha et al. (2018)

The second approach mentioned by Ganzha et al. (2018) is the Central Ontology which is created as a merged ontology from different IoT- and domain ontologies, as seen in figure 42. This ontology is called the Generic Ontology of IoT Platforms (GOIoTP) and acts as a central point for achieving interoperability. The example of four platforms (P, R, S, and T) will now require less Alignment Translators. Each of these platforms has its own ontology (OP, OR, OS, and OT). Between these platforms, the ontologies are first translated to the GOIoTP ontology, and next translated to the target ontology. For example, from platform P to platform S, the ontology of OP will first need to be translated into ontology GOIoTP, which is next translated to ontology OS. The formula for calculating the amount of Alignment Translators in this case is $t = n$, where n is the number of platforms in an ecosystem, and t is the total number of Central Ontology Alignment Translators in an ecosystem. In the example of four platforms, this will include four Central Ontology Alignment Translators.

An advantage of the Central Ontology approach is that it enables scalability. Further, integrating new platforms involves creation of a single pair of alignments with the central ontology. Finally, it requires much less work from the semantic engineer because of the “Single Point of Joining” in the ecosystem. The downside can be the quality of the translations. For every message, two alignments need to be applied, which could result in translation errors. Further, when the central ontology is changed, all the other related ontologies need to be re-generated.

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as a Core Information Model with Extensions. The Core Information Model in this case is the GOIoTP ontology, the extensions are the data sources which are not compliant with the chosen Ontology. The data which is not covered in the GOIoTP ontology is mapped from the source Information Model to the target Information Model.

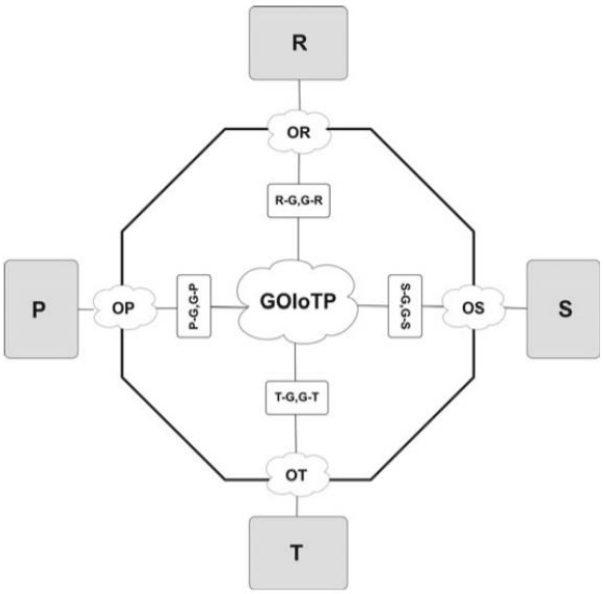

Figure 42: Central Ontology Alignment by Ganzha et al. (2018)

The last approach mentioned by Ganzha et al. (2018) is the Modularized Central Ontology, as seen in figure 43. This approach reduced the risk of the Central Ontology approach of the changes in the central ontology. Suggested is a central ontology, G, which should consist of a core module for common knowledge. This could be a general IoT ontology. This could then be implemented in the specific domain ontologies, all providing separate modules in central ontology G. The example of four platforms (P, R, S, and T) will now require less Alignment Translators than the One-to-One Alignment model, and the same amount of alignment Translators as the Central Ontology model. Each of these platforms has its own ontology (OP, OR, OS, and OT). These ontologies all are part of the domain modules of the core ontology G. For example, ontology OP and ontology OT both represent knowledge from the same domain. This domain is now a component of the core ontology G and is given the name g3.

The first advantage of the Modularized Central Ontology approach is that it provides more scalability than the One-to-One approach, but less scalability than the Central Ontology approach. The quality of the Alignment Translators is better than the Central Ontology approach, but not on the level of the One-to-One approach. This concludes that this Modularized Central Ontology positions himself between these two approaches.

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as the Mapping between Platform-Specific Information Models. There is no Core Information Model in this case, but more a central space of ontology mappings. These mappings are done by finding core concepts matchings in the domains.

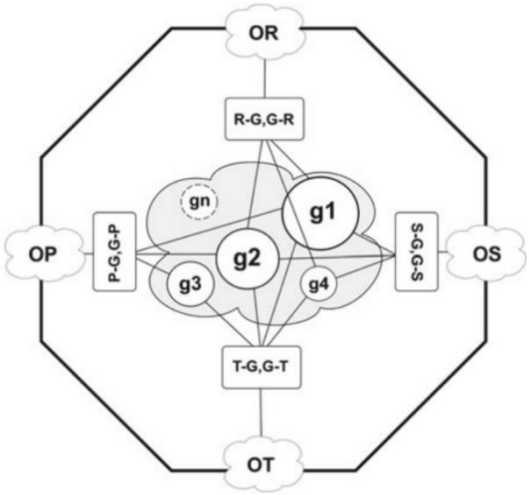

Figure 43: Modularized Central Ontology by Ganzha et al. (2018)

The proposed architecture by Ganzha et al. (2018) follows the Modularized Central Ontology approach. This solution, the IPSM, can be seen in figure 44. The core of the IPSM is the semantic-driven translation of messages. In this IPSM, the input of Producer P will first be translated to Ontology G, followed by a translation of Ontology G to Ontology S, which is next pushed to the Consumer S. This follows the Platform-Specific Information Model Mapping by Vermesan (2018). These mappings are captured in the common ontology G.

Ganzha et al. (2018) mention that it is important to stress that the IPSM is meant to be a generic component, deployed in many instances. Each of these instances join to a specific federation set of platforms, each providing a semantic interoperable infrastructure, fit for its own goals.

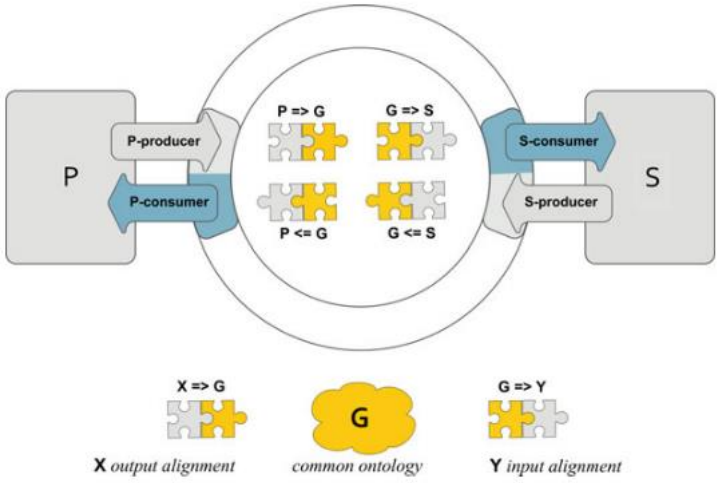

Figure 44: IoT Platform Semantic Mediator (IPSM) by Ganzha et al. (2018)

Integration Manager by Jovanovic et al. (2021)
Jovanovic et al. (2021) described a functional architecture for a Big Data integration platform called Quarry. The Integration Manager (IM) is part of this described architecture and can be seen in figure 45. This IM enables supporting automatic integration of data by generating source-specific metadata if a new data source is registered on the platform. With the source data model and its format, the Wrapper Designer (WD) suggests wrappers which deal with variety in the data sources. This WD also coverts the data into a common unified model.

The Source Schema Extractor (SSE) supports most of the extraction of the schemas of the wrapper. This SSE operates with the help of available data instances to incrementally extract schema information using model-driven transformations. This source schema is expressed in the form of a source graph.

The Global Schema Builder (GSB) supports incremental, bottom-up construction of the global graph of the domain. This GSB relies on semi-automatic alignment and merging of processes, which, supported by the domain experts, produces a vocabulary-oriented knowledge base for the end-users of the system. The techniques used for the semantic alignment are according to Jovanovic et al. the state of the art in ontology alignment. These techniques are Lexical Matching in terms of the Jaccard Index and Wordnet as synonym base (Jovanovic et al., 2021). Based on these matches, the user can accept or reject the suggested match in an intuitive interface.

The Mapping Generator (MG) builds the mapping graphs which are used to relate each variable of a source graph to the global graph. The MG defines the matching source schemas concepts with the global schema concept with “sameAs”. Based on these mapped concepts, possible integration paths can be discovered.

The whole Integration Manager (IM) module builds and maintains the integration graph for resolving semantic conflicts among different data sources, enabling automated data integration and processing (Jovanovic et al., 2021).

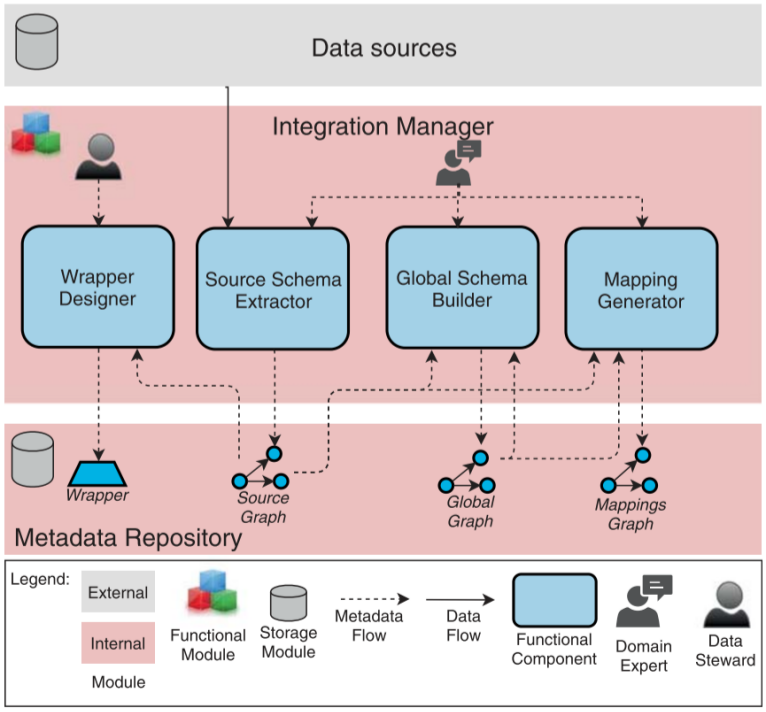

Figure 45: Integration Manager by Jovanovic et al. (2021)

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as the Mapping between Platform-Specific Information Models. The Global Schema can be referred to as a central space of ontology mappings. The mapping of the Source Schemas which do not match the Global Schema can be seen as expanding the Global Graph with specific mappings.

Semantic Matching by Nie et al. (2021)
Nie et al. (2021) provided a design of a big data integration platform based on hybrid hierarchy architecture. This hybrid hierarchy architecture implements Semantic Matching. This architecture

follows Wrappers and Adapters to integrate with different data sources, as can be seen in figure 46. The Wrapper enables the mapping of the relationships between local and global views. The Wrapper in this framework requests the information about the data source pattern, responds to data extraction request, sorts out the results, organizes the results in XML format, and sent it to the integration centre. The data sources are connected to the Wrapper with the help of Adapters. These Adapters are data source specific, meaning different types of data sources require different Adapters. For the Semantic Matching, two different algorithms have been used by Nie et al. (2021). The first algorithm is the Edit Distance. This algorithm is suitable for matching string sequences. The second algorithm is the Jaro-Wrinkler Distance. This algorithm is better at matching string sequences than the Edit Distance algorithm.

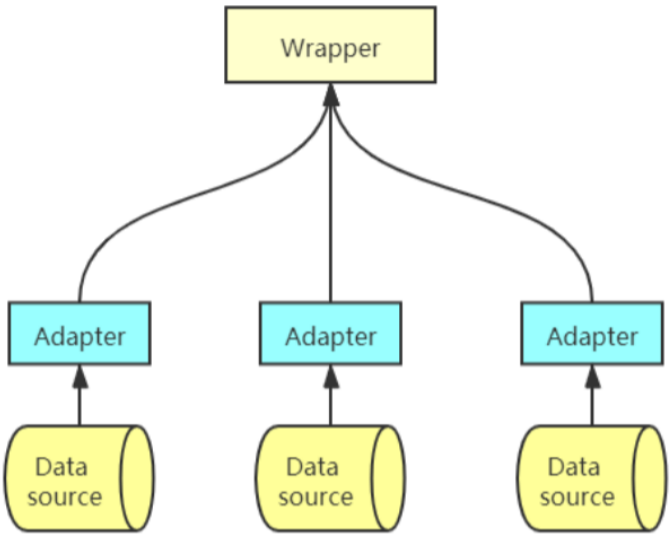

Figure 46: Semantic Matching by Nie et al. (2021)

Looking at the Semantic Interoperability approaches by Vermesan (2018), this approach can be classified as the Mapping between Platform-Specific Information Models. The Global View can be referred to as a central space of ontology mappings. The mapping of the Data Source Schemas which do not match the Global Schema can be seen as expanding the Global Graph with specific mappings.

Implementation Architecture by Balakrishna & Thirumaran (2019)
Balakrishna & Thirumaran (2019) provided an implementation architecture for achieving Semantic Interoperability, as seen in figure 47. This implementation architecture involves six interesting components. The first component is the REST Crawler. This component takes REST resources as input, enabling many types of resources. These Crawlers identify resource formats based on a recursive process of media type discovery and extracts the resources and links presenting the RDF Graph.

The Syntax Extractor updates the by the crawler extracted RDF Graph. The role of this extractor follows five steps for updating the graph, after which it records the made changes in a table.

The Ontology Matcher defines the data of the RDF Graph which is established on semantic key words. The RDF graphs predicates are matched based on these key words, following the ontology alignment algorithm.

The Semantic Reasoner deals with properties and classed which occur in the RDF graphs of the ontologies. The RDF is enriched by adding supplementary predicates. The RDF graph is further refined by the adding of semantic rules.

The Classifier comes helpful when the previous methods proven to be not useful. This Classifier analyses the RDF graph resources with other relation elements with pre-trained approached for RDF graph classification.

The Correlation Analyzer finally adds supplementary links by considering the data change correlations among the relevant IoT resources. The correlation analysis reduces the vague associations between two IoT resources.

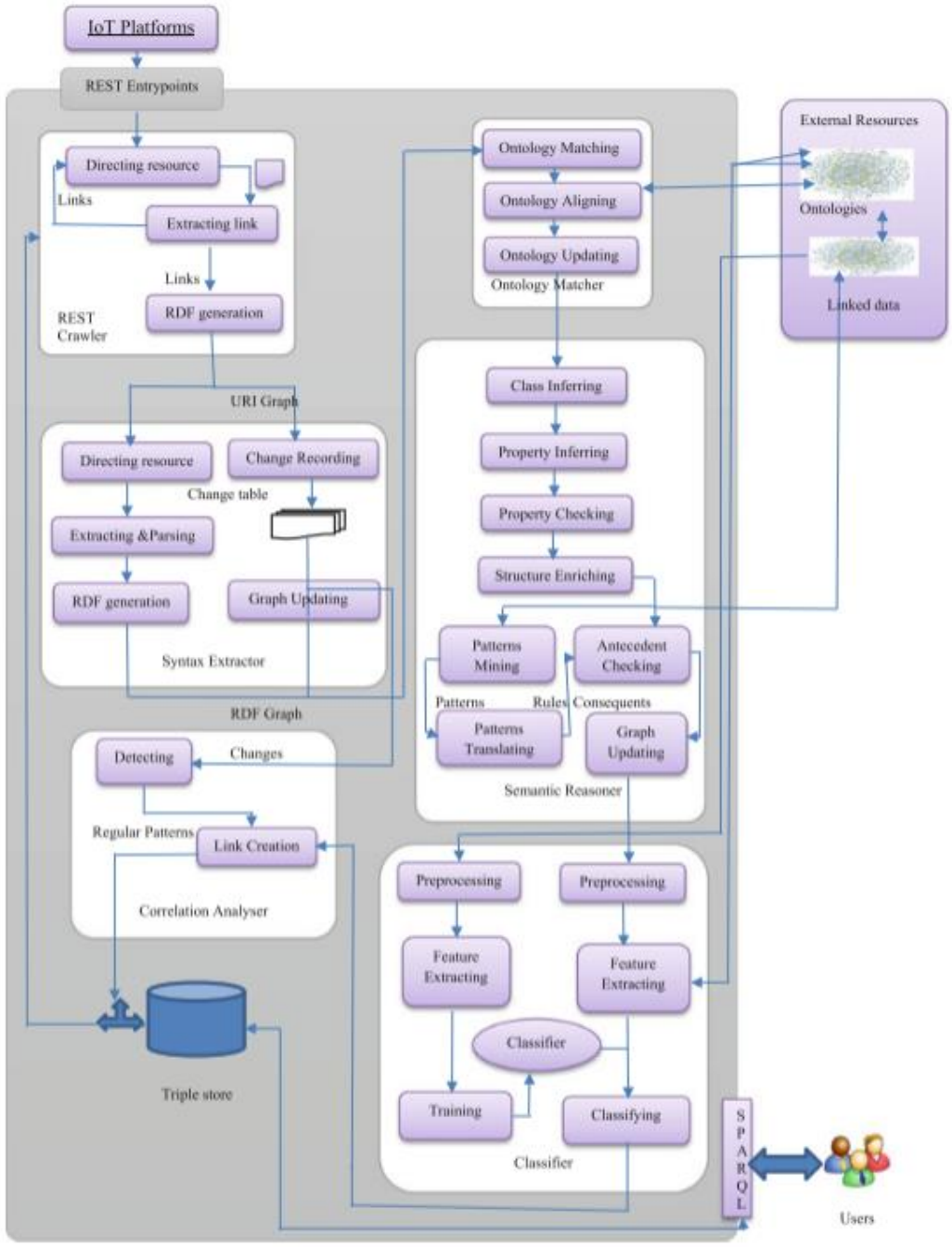

Figure 47: Implementation Architecture by Balakrishna & Thirumaran (2019)

Conclusion

The approach for reaching Semantic Interoperability by Vermesan (2018) have been used to classify different approaches. These approaches either enable Semantic Interoperability by chance, by standardization, or by mapping. The approaches of Vermesan involved the Core Information Model, Multiple Pre-Mapped Core Information Models, Core Information Model with Extensions, Multiple Pre-Mapped Best Practice Information Model, and Mapping between Platform-Specific Information Models.

The Semantic Adapter by Zehnder et al. (2020) could be an approach for achieving Semantic Interoperability. This adapter transforms the data based on the incoming source format and protocol. Further, the MASSIF architecture by Bonte et al. (2017) could be followed for semantically annotating the data based on different contexts. The IoT Platform Semantic Mediator (IPSM) by Ganzha et al. (2018) follows a central ontology for achieving Semantic Interoperability. The incoming data is translated into this central ontology, after which the data is translated to the target ontology.

Further, the Integration Manager by Jovanovic et al**.** (2021) follows a similar approach with a global graph. First the schema is extracted from the data source, this schema is used to extend the global schema, after which the source schema will be mapped using a mappings graph.

The Semantic Matching approach by Nie et al. (2021) uses a Wrapper which incorporates two algorithms to match the source data to the target data source. Finally, the Implementation Architecture for Semantic Interoperability follows an Ontology Matcher and a Semantic Reasoner for matching different IoT data sources, this architecture, and the Integration Manager by Jovanovic et al. (2021), focus more on the initial mapping of the data, while the other architectures emphasize more on the translation between different earlier matched/mapped information models.

A general overview of a Semantic Interoperability architecture, following the ArchiMate ADL, is given in figure 48. This architecture can be seen as a summary of the found architecture. In this architecture, different components of the described solutions have been used. The core of the architecture is based on the Semantic Adapter as described by Zehnder et al. (2020). This Semantic Adapter connects to the Data Sources following the Semantic Matching by Nie et al. (2021). The Integration Manager by Jovanovic et al. (2021) is used for generating the different Semantic Adapters configuration with Source, Global, and Mapping Schemas. These Schemas follow the IoT Platform Semantic Mediator (IPSM) concept as suggested by Ganzha et al. (2018). Further, the Matching Service of Bonte et al. (2017) has been used for Matching the data schemas.

The Implementation Architecture by Balakrishna & Thirumaran (2019) is not used in this Architecture, because it already provides a comprehensive overview of a Semantic Interoperability Architecture, not clearly fitting in the given architecture. The Implementation Architecture by Balakrishna & Thirumaran (2019) is given in the ArchiMate ADL in figure 49. This Implementation Architecture could be plotted on the adopted architecture on the Integration Manager component.

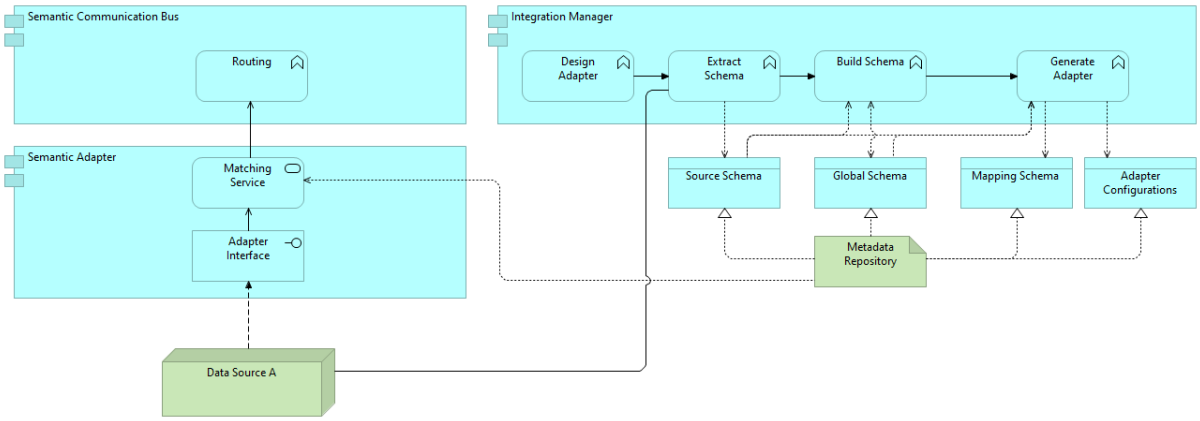

Figure 48: Simplified Semantic Interoperability Architecture following the ArchiMate ADL

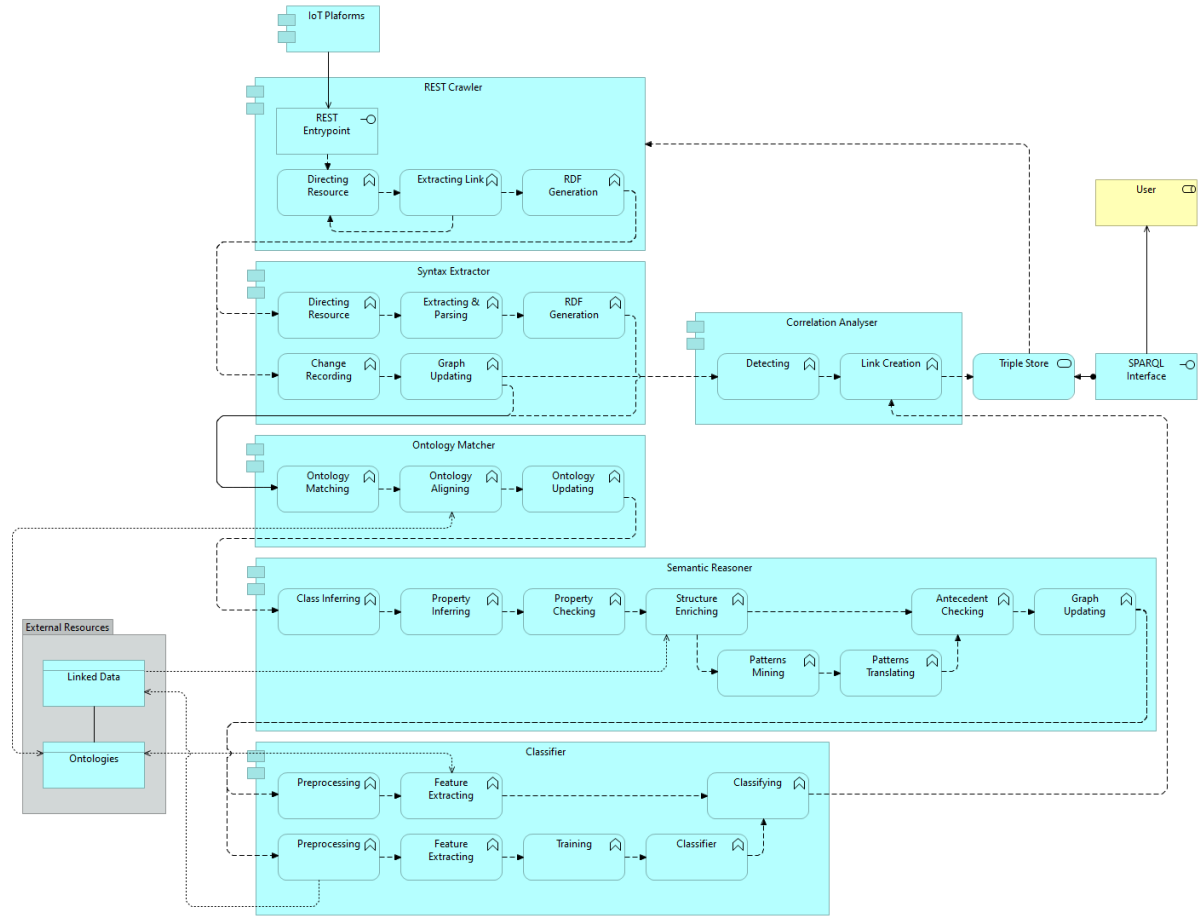

Figure 49: Semantic Interoperability Implementation Architecture following the ArchiMate ADL

To match this Implementation Architecture on the adopted architecture, it could look something like figure 50. This architecture can be seen as a summary of the found architectures in combination with the Implementation Architecture. In this architecture, the functions of the Integration Manager are replaced by the components of the Implementation Architecture. This will enable Semantic Interoperability with the help of different functions, supported by Ontologies and Linked Data.

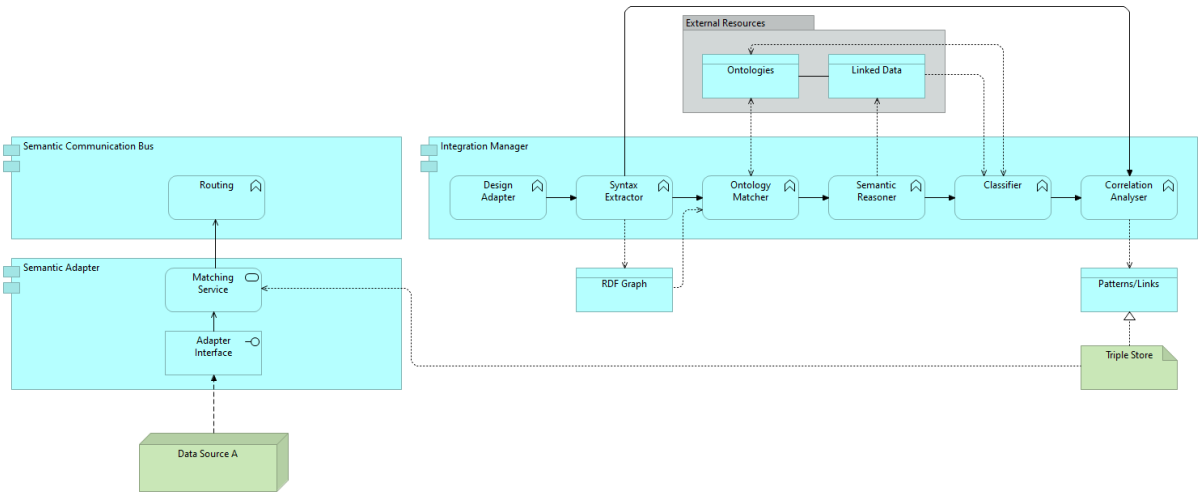

Figure 50: Combined Adopted Semantic Interoperability Architecture following the ArchiMate ADL

5.2 Requirements

A list of the elicited requirements can be found in Appendix A. These requirements have been elicited following the Semi-Structured Interview method of Zowghi & Coulin (2006). This chapter shows the prioritized requirements with the help of MoSCoW method. These requirements will be used to design a treatment. This treatment will follow the qualitative results of the previous chapter, and the requirements according to the stakeholders.

5.2.1 Prioritization of the Requirements & Design Actions

An important next step is to prioritize the stated requirements based on a structured method. Three requirement prioritization methods have been identified. It was chosen to use the MoSCoW method. This method is one of the most used requirement prioritizations. The method prioritizes the stated requirements based on four different categories: Must, Should, Could, and Won't. The "Must haves" are the mandatory requirements, they may not be neglected, or the project will most likely fail. The "Should haves" are great requirements to have, but they are not top priority. The impact on the delivery of the project is low, but they must eventually be implemented. The "Could haves" represent the not essential, but small-scale improving requirements. The "Won't haves" are the requirements with the lowest importance. They do not fit within the objectives of the project or stakeholders and will not be implemented in the final product.

The MoSCoW method is known for its simplicity and Agility for implementations. The downside of this method can be that it lacks the focus on the bigger picture with having blurred lines between the Must Have and Should Have list of requirements.

To make an overall selection for the method for requirements prioritization, the methods were compared with their aspects. As stated, the MoSCoW method provides a simple solution for prioritization of requirements, but the usage of the "Should haves" and "Could Haves" can be confusing. It is chosen to incorporate the MoSCoW method for prioritizing the requirements within this project because its simplicity and agility.

1. Visual Integration Design

This topic consists of the following requirements:

1.1 As an Integration Specialist, I want a visual aid for creating the integrations with different sources and targets, so that I can intuitively design integration workflows, visually connect IoT devices with systems, and streamline the integration configuration process.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a User Interface (UI) which enables the Integration Specialist to intuitively design integration workflows with connecting IoT sources to the target systems and vice versa.
- Design a Business Process which enables the Integration Specialist to create and design integration workflows.

1.2 As an Integration Specialist, I want to be able to create IoT integrations in a low-code manner which supports the most common protocols, so that I can efficiently connect IoT devices to various systems without extensive coding efforts.

This requirement is prioritized as Won't Have, this because of this requirement is a less detailed doubling of requirement 1.3. This specific requirement will therefore not be represented in the Proof of Concept, but a similar requirement will be.

1.3 As a Product Manager, I want the solution to incorporate a No code/low code solution that is model-driven and visually intuitive, so that users can configure and manage integrations without the need for extensive technical expertise.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a User Interface (UI) which enables the Integration Specialists to design the integrations in a Low Code / No Code manner.

1.4 As a Product Manager, I want the solution to remain user-friendly and comprehensible, even with its complexity, so that the solution is accessible to a broad range of users.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a User Interfaces (UI) in a user-friendly manner by following different design patterns for achieving high user-friendliness.

1.5 As a Business User, I want the iPaaS solution to be user-friendly, so that it is accessible and usable by team members across different technical backgrounds.

This requirement is prioritized as Won't Have, this because of this requirement is a less detailed doubling of requirement 1.4. This specific requirement will therefore not be represented in the Proof of Concept, but a similar requirement will be.

2. Dashboard and Monitoring
This topic consists of the following requirements:

2.1 As an Integration Specialist, I want to have clear visibility over the solution with a dashboard that provides graphs and logging about integrations, so that I can monitor the performance and health of integrations in real-time and troubleshoot issues effectively.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a User Interface (UI) which presents a dashboard which shows relevant graphs and logging about the IoT integrations. This is the same dashboard as described in requirement 2.2.

2.2 As a Business User, I want to monitor integration failures and gain insights into the quantity of messages processed successfully, so that I can ensure the smooth operation of our IoT data integrations.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a User Interface (UI) which presents a dashboard which shows relevant information for the Business User about the IoT integrations. This is the same dashboard as described in requirement 2.1.

2.3 As a Business User, I want an alerting mechanism in place, such as email notifications, alerts, the ability to receive SMS or phone calls, so that I can be notified when critical integration issues occur, ensuring immediate response and resolution.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following action:

- Design a Business Process which alerts Business Users when a critical integration issue occurs.

3. Data Mapping and Ontologies:
This topic consists of the following requirements:

3.1 As an Integration Specialist, I want to easily map/match the IoT data to the target data, so that I can ensure seamless data transfer between IoT devices and the intended destinations without data transformation hassles.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept, because it represents one of the core concepts of the research.

This requirement will be satisfied by the following actions:

- Design a User Interface (UI) that allows the Integration Specialist to map and match the IoT data with the target data.
- Design a Business Process that defines how the Integration Specialist will execute the mapping of IoT data to the target data.
- Design the Architecture of the platform that supports the mapping of IoT data in a semantic manner.
- Design a Sequence Diagram of how the system components interact with each other when mapping IoT data to target data.

3.2 As a Product Manager, I want to ensure that users can map their IoT data to the CDM by incorporating ontologies, so that users are able to have more structured and meaningful data integrations.

This requirement is prioritized as Should Have, this because it is important but not vital for the project. If the time is sufficient, this requirement should be implemented in the Proof of Concept.

This requirement will be satisfied by the following actions:

- Design a User Interface (UI) that allows the Integration Specialist to map and match the IoT data with the CDM with the help of ontologies.
- Design a Business Process that defines how the Integration Specialist will execute the mapping of IoT data to the CDM.
- Design the Architecture of a platform that supports the mapping of IoT data to the CDM of an iPaaS.
- Design a Sequence Diagram of how the system components interact with each other while mapping data to the CDM.

3.3 As a Product Manager, I want to ensure that our solution can handle the linking of large data models using ontologies while maintaining manageability, so that users maintain clarity and control over complex integrations, especially with large data volumes.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept, because it represents one of the core concepts of the research.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform that supports the linking of large data model using relevant ontologies.

4. Integration Development and Debugging
This topic consists of the following requirements:

4.1 As a Business User, I need an integration debugger that allows me to examine messages when developing with the integrations, so that it is clearly visible how each message is handled to identify and address issues effectively.

This requirement is prioritized as Could Have, this because it is desirable but not essential for the project. This requirement will not be implemented in the Proof of Concept but needs to be considered while designing the solution.

This requirement will be satisfied by the following actions:

- Design a User Interface (UI) that allows the Integration Specialist to debug the integrations to examine intermediate messages of integration flows.
- Design the Architecture of the platform that supports the debugging of integration flows.

4.2 As a Business User, I want to be able to reuse connections and explore a marketplace of integration solutions, so that I can get inspiration for integrations, leveraging existing resources and best practices to optimize our IoT data integrations.

This requirement is prioritized as Could Have, this because it is desirable but not essential for the project. This requirement will not be implemented in the Proof of Concept but needs to be considered while designing the solution.

This requirement will be satisfied by the following actions:

- Design a User Interface (UI) in which users can inspire integrations based on existing integrations in marketplace function.

- Design the Architecture of the platform that supports a marketplace function for IoT integrations.

4.3 As a Business User, I want that new components can be integrated rapidly and flexibly, so that issues can be resolved as quick as possible.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following actions:

- Design a Business Process which enables rapid integration of new IoT components.
- Design the Architecture of the platform which enables rapid integration of new IoT components.

5. IoT Protocols and Semantic Technologies
This topic consists of the following requirements:

5.1 As a Business User, I require IoT specific protocols (like MQTT in the Pro Glove project), so that seamless communication, data exchange, and interaction with various systems and devices is ensured.

This requirement is prioritized as Should Have, this because it is important but not vital for the project. If the time is sufficient, this requirement should be implemented in the Proof of Concept.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which enables the integration of different IoT specific protocols.
- Design a Sequence Diagram of the interaction between the systems components while integrating with different IoT specific protocols.

5.2 As an Integration Specialist, I want insights into the IoT messaging protocol that sends these messages, the target protocol receiving these messages, and the flow between them, so that I can understand the communication pathways and optimize integrations for efficiency and security.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following actions:

- Design a User Interface (UI) which enables the Integration Specialist to investigate the IoT specific protocols of the source, the target, and the flow between these protocols.
- Design the Architecture of the platform which enables the investigation of the IoT specific protocols of the source, the target, and the flow between these protocols.
- Design a Business Process which enables the investigation of the IoT specific protocols of the source, the target, and the flow between these protocols.

5.3 As a Product Manager, I want the solution to support semantics for sensor data coming from IoT hubs and other applications, so that I can enhance the capabilities of the solution.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept, because it represents one of the core concepts of the research.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which enables the support for semantic sensor data coming from IoT hubs and other applications.
- Design a Sequence Diagram of the interaction between the system components enabling the support for semantic sensor data coming from IoT hubs and other applications.

5.4 As a Product Manager, I want the solution to be compatible with semantic technologies like JSON-LD, so that users can expose or consume semantic data and link it to their data model seamlessly.

This requirement is prioritized as Should Have, this because it is important but not vital for the project. If the time is sufficient, this requirement should be implemented in the Proof of Concept.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which supports JSON-LD for integration IoT sources.
- Design a Sequence Diagram of the interaction between the system components enabling the support for JSON-LD.

6. Scalability and Reliability
This topic consists of the following requirements:

6.1 As a Business User, I want the iPaaS architecture to be scalable, so that I can anticipate on, and handle increased data loads efficiently.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which is scalable by applying best practices for achieving scalability for such platform.

6.2 As a Business User, I want the integrations to have an appropriate reliability and availability for specific use cases, so that the clients working with the product experience no errors, ensuring a high level of service.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which achieves a high reliability and availability by applying best practices for achieving these for such platform.

7. Compatibility
This topic consists of the following requirement:

7.1 As a Product Manager, I want the solution to be compatible with the definition and both run-time and design-time environments of eMagiz, so that seamless functionality throughout the entire integration development and execution process is ensured.

This requirement is prioritized as Must Have, this because it is critical for the success of the project. This requirement may not be neglected when building the Proof of Concept. In agreement with the supervisor, this requirement has been classified as one of the logical requirements of the treatment because it is already captured in a general iPaaS architecture.

This requirement will be satisfied by the following actions:

- Design the Architecture of the platform which is compatible with the definitions, run-time environment, and design-time environment of eMagiz.

5.3 Solution Design

The solution design consists of five different design aspects. First, the functionalities describe the behaviour of the designed treatment. Next, the processes show the flow between different aspects of the treatment and the stakeholders. The architectures show how different technical components of the treatment interact with each other. The sequence diagrams further show the interaction between the components in the treatment. The user interfaces finally give a better view of what the treatment could graphically look like. The validation of the solution design is discussed in the evaluation chapter.

The designed treatment can be described as a solution which enables Semantic Interoperability for IoT integrations on iPaaS with the help of five concrete components. The first component is the Source Schema Extractor. This component extracts the schema from the provided source system. This component consists of an MQTT Proxy, a module which subscribes to an MQTT topic, and a module which extracts the source schema from an incoming MQTT payload. The concept of the MQTT Proxy is based on Hettig (2021) as identified in the qualitative findings of the research. The Source Schema Extractor is based on the Schema Extractor of the Integration Manager by Jovanovic et al. (2021) as identified in the qualitative findings of the research.

The second component is the Ontology-Schema Matcher. This component matches the extracted Source Schema to the defined Global Ontology. This matcher is based on the concept of Semantic Matching proposed by Nie et al. (2021) as identified in the qualitative findings of the research.

The third component is the Ontology Searcher. This component enables the Integration Specialist to search relevant domain extensions for the Global Ontology. The Searcher is based on the Ontology Matcher & Semantic Reasoner of the Implementation Architecture by Balakrishna & Thirumaran (2019) as identified in the qualitative findings of the research.

The fourth component is the Global Ontology. This Global Ontology consists of a Core IoT Ontology and can be extended by several Domain Specific Ontologies. The concept of this component is derived from the IoT Platform Semantic Mediator (IPSM) by Ganzha et al. (2018) as identified in the qualitative findings of the research.

The fifth component is the Semantic Adapter Generator. This component is based on the Integration Manager by Jovanovic et al. (2021) as identified in the qualitative findings of the research. This Generator generates the Semantic Adapters for the specified IoT Integrations. This generated Semantic Adapter is based on the Semantic Adapter by Zehnder et al. (2020) as identified in the qualitative findings of the research.

5.3.1 Design Methods

5.3.1.1 Solution Design

The solution design for this project consists of different design aspects. These aspects are Functionalities, Architectures, Processes, and Interfaces.

Functionalities

The functionalities will be designed in User Stories, which is following the same method as the elicited requirements. The functionalities will be a more detailed specification of the functionalities of the solution, based on the given requirements.

Architectures

For designing the architectures, the ArchiMate Description Language will be used. The consideration for this ADL is described in the qualitative findings section of this report.

Processes

For the Processes, two different components can be distinguished. The first component being the business processes. This incorporates the design of the business specific processes, involving the relevant actors and events within these processes. The method which will be used for designing the processes is the Business Process Modelling Notation (BPMN) method. This method has been developed by the Business Process Management Initiative (BPMI) in 2004 (White, 2004). BPMN bridges the gap between business process design and process implementation. Further, BPMN provides a Business Process Diagram (BPD) based on a flowcharting technique. An example process following the BPMN method is given in figure 51.

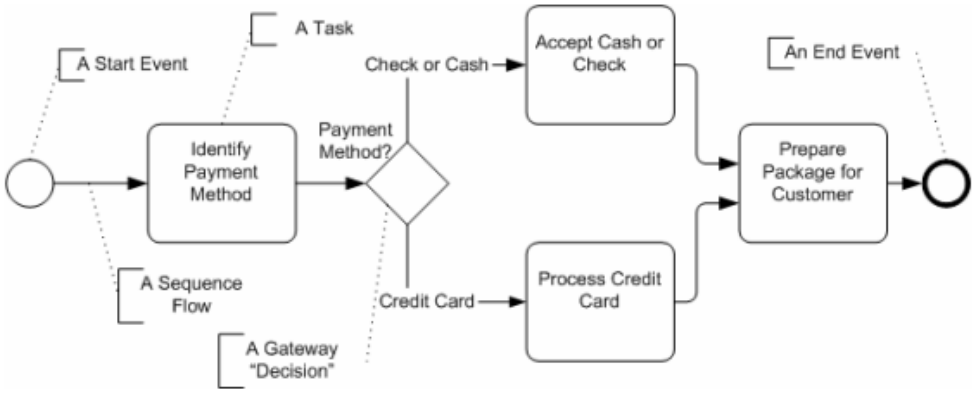

Figure 51: Example Business Process Following the Business Process Modelling Notation (BPMN) by White (2004)

The second component is the interaction between the systems, which is especially relevant for an iPaaS architecture. The method chosen for these interactions is the Sequence Diagram. This Sequence Diagram is part of the Unified Modelling Language (UML) for Object-Oriented Development as introduced by Booch et al. (1996). This sequence diagram shows process interactions of systems within time sequence. An example Sequence Diagram can be seen in figure 52.

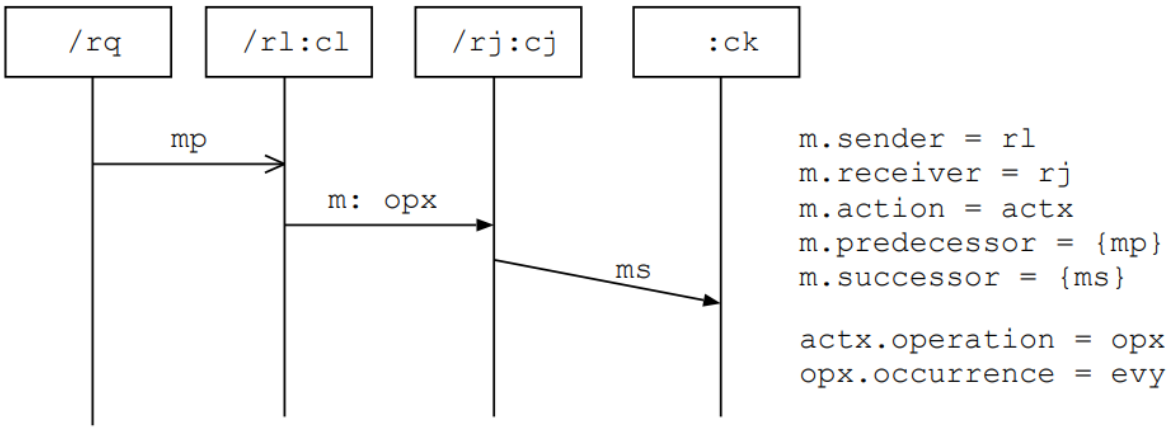

Figure 52: Example Sequence Diagram by Bernardi et al. (2002)

Interfaces

The interfaces involve the screens the users, in this case the Integration Specialists, gets to work with. This user interface is one of the most important elements of a computer-based system. If this interface is poorly designed, the adoption the solution may fail (Sridevi, 2014).

Sridevi (2014) provided three important principles for effectively designing user interfaces. The first principle is to place the user in control. This involves that the interface should never require the user to interact directly "in" a machine (like a command line), but rather in an intuitive interface. The second principle is to reduce the memory load of the user. This involves the reduction of the amount of information the user needs to remember. The crucial information always needs to present in the interface to enable the user to optimally use the interface. The final principle is to make the interface consistent. This involves designing a consistent interface to not confuse the users, meaning a consistent graphical and experience design needs to be implemented across the solution.

For designing the interface, wireframes will be created. Gudoniene et al. (2023) mention that wireframes are an important part of the User Experience (UX) design process. Further, wireframes are described as a representation of a page layout or drawing, with the purpose to visually transmit the initial design idea of a website or application. A wireframe serves as a blueprint or framework for the User Interface, allowing designers and stakeholders to understand the final product before it is built (Gudoniene et al., 2023). The main elements of a wireframe can be seen in figure 53.

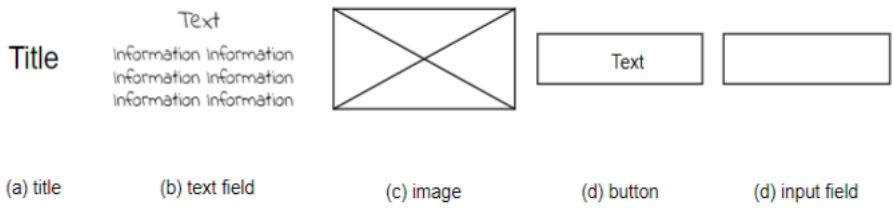

Figure 53: Main Elements of a Wireframe by Gudoniene et al. (2023)

5.3.2 Design Choices

Different design choices have been made as part of the Solution Design of this project. This Solution Design is validated by the company supervisor and the CTO of the company.

There are many different relevant protocols and formats in the field of Semantic Interoperability and IoT architectures. Below an overview of the selected relevant protocols and formats is given.

IoT Communication Protocol

The selected IoT protocol within this Solution Design is MQTT. It is chosen to use MQTT because as of today, MQTT seems to be leading the way in IoT applications (Priyadarshi & Behura, 2018). Farhan et al. (2022) further mention that MQTT is one of the most used protocols within the field of IoT.

iPaaS Event Streaming

Because most of the IoT integrations follow the Event Streaming integration pattern, it was chosen to select Apache Kafka as base solution within the designs. Apache Kafka is a popular solution for incorporating Event Streaming on an iPaaS architecture (Chatt, 2019).

Connecting the IoT Protocol to the Event Streaming protocol

To connect an MQTT client to Apache Kafka, two options have been identified by Hettig (2021). A choice needs to be made between these two options. The first option is using an MQTT Broker which connects to the Kafka Broker with the help of an MQTT Broker source connector. The second option is the use of an MQTT Proxy. This option connects the MQTT Clients to the Kafka Broker without the use of an MQTT Broker. The Proxy forwards the incoming payload to a pre-defined Kafka Topic. For the Solution Design, it was chosen to use the MQTT Proxy solution. This solution is more cost effective, because not the whole MQTT-broker is needed.

Integration Lifecycle Management

The integration Lifecycle Management approach employed in the solution design follows the Integration Lifecycle Management (ILM) method employed in the iPaaS platform of the sponsor. This ILM method consists of five different phases. These phases can be seen as an adaption of the B-model as proposed by Birell & Ould (1988). In this B-model, a lifecycle with different phases for developing software is given. An overview of this B-model can be seen in figure 54.

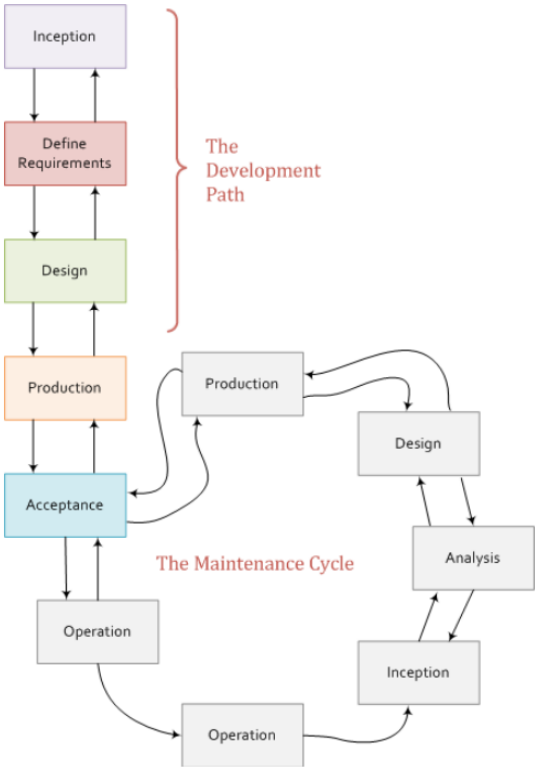

Figure 54: B-model Development Lifecycle by Ruparelia (2010)

The first phase is the Capture phase. In this phase, the requirements to the integration are captured. These requirements give a high-level overview of the integrations which need to be developed. This can be seen as the “Define Requirements” phase as mentioned by Birell & Ould (1988).

The second phase is the Design phase. In this phase, the mappings between the source systems and the data model of the integration platform are designed. This can be seen as the “Design” phase as mentioned by Birell & Ould (1988).

The third phase is the Create phase. In this phase, the routing of the message during the integration is created. This can be seen as the “Production” phase as mentioned by Birell & Ould (1988).

The fourth phase is the Deploy phase. In this phase, the created integrations are deployed into the production environment of the client. This can be seen as the “Acceptance” phase as mentioned by Birell & Ould (1988).

The final phase is the Manage phase. In this phase, the client can monitor and/or improve its deployed integrations. This can be seen as the “Maintenance Cycle” as mentioned by Birell & Ould (1988). This Maintenance Cycle consists of iterations on the previous mentioned phases.

5.3.3 Functionalities of the artifact

For stating the functionalities of the artifact, first the requirements have been elicited. These requirements are obtained from interviews with the relevant stakeholders identified in the stakeholder analysis. The requirements can be found in the Requirements section of this chapter.

The final set of functionalities for the solution in the User Story format is described in this section. These functionalities are modelled as Epics, followed by a set of User Stories. These are the functionalities which will be implemented within the Proof of Concept (prototype) of the final project.

1. Source Schema Extractor:
1.1. As an Integration Specialist, I want to effortlessly extract the source schema of an IoT gateway or application so that I can seamlessly integrate it with other systems connected to the iPaaS.

1.2. As an Integration Specialist, I want a visually intuitive representation of the extracted source schema of an IoT gateway or application so that I can get a clear understanding of its structure within the iPaaS environment.

1.3. As an Integration Specialist, I want the capability to modify the extracted source schema of an IoT gateway or application so that I can rectify any discrepancies or inaccuracies in the schema representation.

2. Schema-Ontology Matcher:
2.1. As an Integration Specialist, I want the source schema to be automatically aligned with the Global Ontology of the integration environment so that there is a seamless matching of the source system's schema with the iPaaS semantics.

2.2. As an Integration Specialist, I want a visual representation of the mapping between the source schema and the Global Ontology so that I can understand and validate the connection between these systems.

2.3. As an Integration Specialist, I want the ability to edit the mapping of the source schema to the Global Ontology so that I can ensure accurate alignment as per my comprehension.

3. Ontology Searcher:
3.1. As an Integration Specialist, I want to efficiently search for domain ontologies to enrich the Global Ontology so that I can incorporate the most contextually relevant semantics into the iPaaS.

3.2. As an Integration Specialist, I want the selected domain ontology to seamlessly integrate with the Global Ontology, ensuring automatic alignment with the iPaaS semantics.

3.3. As an Integration Specialist, I want the functionality to modify the mapping between the discovered domain ontologies and the Global Ontology so that I can ensure precise integration and alignment between these semantic elements.

4. Global Ontology:
4.1. As an Integration Specialist, I want a comprehensive visual overview of the Global Ontology so that I can gain a clear understanding of the semantic structure within the iPaaS environment.

4.2. As an Integration Specialist, I want the ability to edit and enhance the Global Ontology so that I can refine the semantic framework within the iPaaS environment.

5. Semantic Adapter:
5.1. As an Integration Specialist, I want an automated generation of a semantic adapter based on the mapped source schema to the Global Ontology so that I can smoothly deploy IoT integrations on the iPaaS's Run-time.

6. Low-Code:
6.1. As an Integration Specialist, I want to create IoT integration flows in a Low-Code manner so that I can easily and comprehensively manage and understand IoT integrations within the iPaaS.

7. Dashboard:
7.1. As an Integration Specialist, I want a dashboard which visualizes indicators of the IoT integrations, so that I can get a clear overview of the performance of the integration environment on the iPaaS Run-Time for effective monitoring and management.

7.2. As a Business User, I want a dashboard which visualizes indicators of the IoT integrations, so that I can gain a clear overview of the amount of traffic which crosses the iPaaS, aiding in better decision-making and insights into operations.

5.3.4 Designed Processes

The business processes are designed based on the given requirements. The business processes are modelled in the Business Process Modelling Notation (BPMN) within the BPMN.io modeller and are presented as described in the methodology section of this chapter. The designed Processes are provided in the Appendix of this report. The defined identifier, in this case BP[identifier of the design] refers to the corresponding Business Process in Appendix B.

BP1. Design-Time Process
The first designed process is based on a current Integration Lifecycle which consists of six phases. These phases are 1) Capture the Integration Requirements, 2) Design the Integration, 3) Create the Integration, 4) Deploy the Integrations, 5) Manage Integration, and 6) Improve the Integration. These are existing phases of an iPaaS platform. To enable such an iPaaS solution to incorporate Semantic Interoperability, an extension of the Design Phase is needed. Below, a new event is introduced called the "Create Semantic Adapter" event.

The first step in this event is to extract the source schema from the IoT gateway. This schema is then compared with a global ontology, which is unique for each integration environment but based on the same core IoT ontology.

If there is a complete match between the source schema and the global ontology, the integration specialist assesses the match. If the match is not perfect, a relevant ontology is retrieved from the semantic web. This retrieved ontology is then merged with the global ontology, and the integration specialist assesses the match again.

If the match is still not satisfactory, the process loops back to the schema matching step. If the match is successful, the global ontology is updated if necessary. After this, a Semantic Adapter is generated.

The Semantic Adapter is essentially a set of transformation rules used by the Kafka Event Processor. These rules translate the incoming payload to match the global ontology of the environment, preferably in JSON-LD format.

This process ensures that the data from different IoT gateways can be integrated seamlessly into the platform, enhancing its interoperability and efficiency. It is a smart design that leverages the power of semantic web technologies to improve IoT integrations on iPaaS platforms.

BP2. Run-Time Process
The second designed process is based on the Event Streaming pattern of Apache Kafka. This uses a Kafka Source Connecter, Kafka Stream Processor, and a Kafka Sink Connector to enable systems to publish and subscribe to the Kafka topics.

This process starts with an IoT Gateway that captures sensor data. This data is collected either when there is a change in the environment or at regular intervals. Once the data is captured, it is published to a topic on the MQTT proxy.

The MQTT proxy then forwards this payload to the Kafka Broker on the Kafka Source Connector. The Kafka Stream Processor first transforms the payload to the Global Ontology, following the Transformation Rules generated by the Semantic Adapter. After this transformation, the Stream Processor can execute other defined processing tasks on the payload.

Once the Stream Processor has finished processing, the payload is sent to the subscribers via the Sink Connectors of the Kafka Broker. All systems which are subscribed to the specific topic then receive the payload.

In terms of error handling, all components of the system, including the Kafka MQTT proxy, the Kafka Connector (Source & Sink), and the Kafka Event Stream Processor, log any errors they encounter. After these errors are logged, the Run-Time saves the messages in a database. The Business User is then notified of the error through a Dashboard Notification, SMS, or Mail.

5.3.5 Designed Architectures

The architectures are designed based on the given User Stories and Processes. The architectures are modelled in the ArchiMate Architecture Description Language within the Archi Modeller as described in the methodology section of this chapter. The designed Architectures are provided in the Appendix of this report. The defined identifier, in this case A[identifier of the design] refers to the corresponding Architecture in Appendix B.

A1. Design-Time
The proposed Design-Time architecture is an expansion of an existing iPaaS platform, designed to enhance the integration and interoperability of IoT gateways. The architecture is based on the

Capture, Design, Create, Deploy, Manage, and Improve phases, with a particular focus on the Design phase.

In the Design phase, the Integration Modeller application component is enhanced with several new application functions. These include the Schema Extractor, Schema-Ontology Matcher, Ontology Searcher, Global Ontology Updater, and Semantic Adapter Generator.

The Schema Extractor is responsible for extracting the source schema of the IoT gateway. This extracted schema is then matched to the Global Ontology by the Schema-Ontology Matcher. The Ontology Searcher then searches the Semantic Web for relevant Best-Practice ontologies to expand the global ontology. Once these ontologies are identified, the Global Ontology Updater updates the global ontology. Finally, the Semantic Adapter Generator generates the Semantic Adapter, which consists of the Transformation Rules and Connectors that enable Kafka to semantically interoperate IoT sources.

This Design-Time architecture will be developed and run on Mendix in a Virtual Private Cloud. This ensures a secure and scalable environment for the architecture to operate in.

A2. Run-Time
The proposed Run-Time architecture is an expansion of an existing iPaaS platform which uses Kafka as Event Streaming Broker. The architecture is following the Event Streaming pattern of Kafka which involves a Source Connector, Event Stream Processor, and a Sink Connector.

The external IoT gateway application component publishes data to a topic on the Kafka MQTT Proxy application component using the MQTT protocol. The payload is next forwarded to the relevant Kafka Topic on the Kafka Source Connector (Topic In) interface of the Kafka Broker.

After the payload is received by the Kafka Broker, it is processed by the Kafka Event Stream Processor application service, which, based on the transformation rules, processes the payload. After the payload is successfully processed, it will be published to the subscribers by the Kafka Sink Connector (Topic Out) interface. These subscribers can be a variety of solutions, like another Application, or another Integration on the iPaaS environment.

This Run-Time architecture will be run on an Amazon Web Service (AWS) Managed Streaming Service for Apache Kafka (MSK) in a Virtual Private Cloud. This ensures a secure and scalable environment for the architecture to operate in. This AWS MSK will use the generated configurations to deploy the comprehensive solution. The Kafka Broker and Kafka Connect components will be running on this AWS MSK Virtual Private Cloud. The Kafka MQTT Proxy and the Kafka Stream Processor will be hosted on AWS Elastic Compute Cloud (EC2) instances.

5.3.6 Designed Sequence Diagrams

The sequence diagrams are designed based on the given User Stories, processes, and architectures. The UML-notation has been used for creating the designs, with the help of the online tool Sequencediagram.org. The designed Sequence Diagrams are provided in the Appendix of this report. The defined identifier, in this case SD[identifier of the design] refers to the corresponding Sequence Diagram in Appendix B.

SD1. Design-Time
The proposed Design-Time sequence diagram follows the interaction between different actors and system components.

The sequence starts with the Business User presenting a Business Initiative for an IoT Integration to the Integration Specialist. The specialist takes charge of the initiative and engages with the iPaaS for the development of the IoT integration.

Upon activation, the iPaaS communicates with the IoT Source to Extract the Source Schema, obtaining the Source Schema required for integration. Next, the iPaaS proceeds to Match the Source Schema with the Global Ontology, interacting with the Semantic Web to Crawl Relevant domain Ontologies. The identified Relevant Ontologies are then Merged and Matched with the Global Ontology within the iPaaS environment.

After this, the Environment Deployer is activated to Deploy Environments and Generate Run-Time Configurations. This deployment involves a Test Environment, ensuring the Integration Environment can be tested by the Integrations Specialist.

Continuing this sequence, another segment includes the deployment of an Acceptation Environment. Here, the Business User conducts an acceptation test within the Accepting Integration Environment, leading to the Accepted status communicated back to the Environment Deployer.

Subsequently, the final sequence involves the Deployment of the Production Environment conducted by the Environment Deployer, resulting in the successful completion of the Design-Time sequence.

This sequence diagram defines the step-by-step progression of the Design-Time workflow, illustrating the orchestrated collaboration between stakeholders and technical components for the successful implementation of IoT integration using the iPaaS platform.

SD2. Run-Time
This Run-Time sequence diagram describes the interactions between the different system components during the Run-Time of the iPaaS, involving IoT devices and data processing through Kafka. Before the main branch of this sequence is executed, first the Consumers must subscribe to the Kafka Sink Connector (Topic Out) to receive the data in the end.

The sequence starts with the IoT Device publishing sensor data with the help of one of the multiple communication protocols (Zigbee, Z-Wave, BLE, WiFi) to the IoT Gateway.

Next, the IoT Gateway communicates with the Kafka MQTT Proxy to publish the sensor data to the MQTT Topic, facilitating the forwarding of data to the Kafka Source Connector (Topic In).

The Kafka Source Connector then processes the received payload and communicates with the Kafka Stream Processor to initiate the payload processing. This involves transforming the data to comply with the Global Ontology, and executing other specified transformations to optimize the data before it is pushed to the subscribers.

Upon completion of processing, the Kafka Stream Processor sends the processed data back to the Kafka Sink Connector (Topic Out), which subsequently publishes this processed data to the waiting Consumers.

This data flow and processing sequence within the Run-Time environment ensures the seamless processing and transformation of sensor data, conforming to a defined ontology, and delivering the refined data to the designated consumers.

5.3.7 Designed User Interfaces

The user interfaces are designed based on the given User Stories, processes, architectures, and sequence diagrams. The user interfaces are designed using wireframes within the Canva.com design

tool and are provided in the Appendix of this report. The defined identifier, in this case UI[identifier of the design] refers to the corresponding User Interface in Appendix B.

UI1. Capture – Requirements Designer:
The first wireframe, the Requirements Designer, shows the available integration solutions in the left panel and visually represents the iPaaS in the larger right panel. Here, the Integration Specialist can drag the IoT Gateway onto the iPaaS canvas and create integration by drawing lines between the solutions and the iPaaS.

UI2. Capture – Edit IoT Gateway:
Upon opening the IoT Gateway solution, the Integration Specialist can modify specific elements like Gateway names, message formats, the MQTT proxy host, topics & devices, and the task state of the integration.

UI3. Design – Solution Designer:
Following the capture, the Integration Specialist moves to design the integration. This stage involves defining the source schema of the IoT gateway, particularly through the Extract Schema function, allowing customization and schema creation.

UI4. Design – Waiting for incoming payload:
The Extract Schema function prompts the iPaaS to connect to the designated MQTT proxy, receiving payloads from the IoT gateway, thereby initiating the schema extraction process.

UI5. Design – Received Data:
Post-receiving the payload, the Integration Specialist can view and edit the payload in JSON format based on their expertise. This modified schema needs to be saved for further extraction.

UI6. Design – Extracted Schema:
Upon saving the payload the schema is extracted. This schema is visually represented as a model with entities, attributes, and relationships. The Specialist can further edit this model to match their knowledge, after which it must be mapped to the Global Ontology.

UI7. Design – Schema Mapping:
Mapping the schema to the Global Ontology creates an overview displaying entities and their attributes, in relation to their corresponding classes. This matching process is editable, and if the Global Ontology lacks coverage, a Domain Ontology can be added.

UI8. Design – Add Domain Ontology:
Adding a Domain Ontology involves selecting relevant ontologies, aligning them with the Global Ontology, and updating this Global Ontology. The source schema will be rematched to the updated Global Ontology, which needs approval or further editing by the Integration Specialist.

UI9. Design – View Global Ontology:
Upon approval, the Global Ontology is visible, presenting any extensions made by the Domain Ontologies. This visual representation includes the Core IoT Ontology extended with relevant Domain Ontologies.

Once the schema-to-Ontology mapping is approved, a Semantic Adapter is generated in the background. This adapter includes transformation rules for IoT schemas into JSON-LD, configured with MQTT Proxy settings. Additionally, data transformations based on CDM/ Event Streaming Data Model (ESDM) specifications from the Global Ontology are also facilitated.

UI10. Create – Processes:
Finally, the created transformations can be used in the Integration Flow designer of an iPaaS portal. Two types of transformations are generated, the Source to JSON-LD, and the Source to CDM/ESDM. The first one can be used to expose the IoT data in a semantic manner to other systems in the iPaaS environment. The other transformation can be used to match the incoming data to the CDM/ESDM of the integration environment, after which it can be used in other integration flows or applications.

5.4 Conclusions

iPaaS Architectures
To answer the issued research questions, first, a general architecture for iPaaS was needed. To get a general architecture of iPaaS, different sources have been combined. For iPaaS and integration platforms, the layered model is a common architectural pattern. This layered model generally consists of three horizontal layers: the Data layer, the Application layer, and the Presentation layer.

The Data layer is responsible for receiving, transforming, and routing data. It includes components like the Enterprise Service Bus, Database, Conversions, Interface Adapters, and potentially the Data Fusion Service and Data Push Service. It also involves the extraction and reconfiguration of data for consumption and may include components from the Data Plane such as the Proxy, Gateway, CI/CD pipeline, and System of Records.

The Application layer manages various components, executes processes, and diagnoses the platform. It includes elements like the Platform Manager, Data Source Manager, Analytics & Decision Support, Computations, Monitoring service, Workflow Execution service, and the Web Service Management Centre, and is seen in various architectures as a combination of other layers or services. It does not include Data Fusion & Data Push services, which belong to the Data layer.

The Presentation layer in most platform architectures provides an intuitive interface for managing the platform, creating integrations, and configuring flows. It involves various components such as the User Interface, Dashboard, Platform Management Tool, Services, and elements of Code/Low-Code/No-Code, Connectors, Policy Enforcements, and an API Marketplace.

A general iPaaS architecture has been visualized consisting of three horizontal layers and one vertical layer. This layered model provides a structured and systematic approach to understanding and implementing iPaaS architectures.

IoT in iPaaS Architectures
The first research question of this project answered by this chapter, is stated below. This question is answered based on research in integrating IoT in iPaaS architectures.

(SQ1) - How can IoT data integrations be incorporated in iPaaS architectures?

The architecture of IoT can be broadly categorized into three horizontal layers: the Device layer, the Integration layer, and the Presentation layer. The Device layer consists of the physical IoT devices, including Sensors, Actuators, and the Processors/IoT Gateways. The Integration layer serves as a bridge between the Device layer and the subsequent layer, processing events or topics. The Presentation layer encompasses the end-applications and user interfaces.

Notably, the Fog-Enabled IoT platform Architecture by Asemani et al. (2019) includes a Fog layer within the Device layer and the RA for IoT by Sobin (2020) incorporates an Aggregation/Bus layer within the Integration layer.

Furthermore, a Security component is present in both the RA for IoT by Sobin (2020) and the Fog-Enabled IoT platform Architecture by Asemani et al. (2019). This Security component is absent in the IoT RA by Guth et al. (2016). This highlights the need for a Security component to ensure a comprehensive IoT architecture.

Therefore, a general IoT architecture has been visualized consisting of three horizontal layers and one vertical layer. This layered model provides a structured and systematic approach to understanding and implementing IoT architectures.

Further, as described by Sobin (2020), the solution needs to communicate with the devices with the help of different communication protocols. Relevant protocols for IoT are MQTT, CoAP, DDS, AMQP, and XMPP. Different implementation solutions have been identified for each of these protocols. The MQTT protocol has been identified as one of the most suitable solutions to integrate IoT gateways with iPaaS architectures.

Semantic Interoperability in iPaaS Architectures

The second research question of this project, which is answered by this chapter, is stated below. This question is answered based on research in achieving Semantic Interoperability in iPaaS architectures.

(SQ2) - How can Semantic Interoperability be incorporated in iPaaS architectures?

There are different methods to achieve Semantic Interoperability in iPaaS architectures. These methods range from service and data viewpoint classifications to semantic adapters, integration managers, and semantic matching techniques. Each approach has its unique strengths and focuses, whether it is initial data mapping or translation between previously matched information models.

The proposed Semantic Interoperability architecture combines these different methods, leveraging the strengths of each while maintaining a cohesive structure. It is based on the Semantic Adapter by Zehnder et al. (2020), incorporates Semantic Matching by Nie et al. (2021), utilizes the Integration Manager by Jovanovic et al. (2021) for generating configurations, and employs the IoT Platform Semantic Mediator (IPSM) concept by Ganzha et al. (2018) for the schemas. The Matching Service of Bonte et al. (2017) is also integrated for data schema matching.

While the Implementation Architecture by Balakrishna & Thirumaran (2019) provides a comprehensive overview, it is not included in this architecture due to its extensive scope. However, it could be incorporated into the Integration Manager component if needed. These various approaches underscore the complexity of achieving Semantic Interoperability, highlighting the need for continued research and innovation in this field.

Solution Design

The third research question of this project, which is answered by this chapter, is stated below. This question is answered based on the results of the first two research questions, and on the input of the relevant stakeholders. The answer on the research questions is mainly given by the designed architecture, consisting of the two environments.

(SQ3) - What are the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across different systems?

The requirements for the Solution Design are based on the input of different stakeholders which are elicited with help of semi-structured interviews. The requirements are stated in the User Story format, after which they were prioritized using the MoSCoW method.

With the input of the User Stories and research results, different design actions have been stated. These actions resulted in the Solution Design consisting of different Processes, Architectures, Sequence Diagrams, and User Interfaces.

The solution design can be concluded as two environments providing a complete solution for the problem. The Design-Time environment consists of different phases. The Capture, Design, Create, and Deploy phase. Within the Design phase, the Semantic Adapters are created with the help of the Source Schema Extractor, Ontology Matcher, Ontology Searcher, and the Semantic Adapter Generator. This Generator generates the transformation rules, and the configurations for the Run-Time environment.

The Run-Time environment consists of an MQTT Proxy, which forwards the incoming IoT payload to a pre-defined topic on the Apache Kafka Source Connector. This Connector enables the Kafka Stream Processor to execute payload transformations. One of these transformations is transforming the incoming source payload to the Global Ontology of the integration project in JSON-LD format. This Global Ontology consists of a Core IoT Ontology, extended with project specific Domain Ontologies. With the data now in JSON-LD format, the with context enriched data can be consumed by systems which are enabled by Semantic Interoperability. The data can also be transformed to the Canonical Data Model of the integration project. The Kafka Sink Connector finally delivers the transformed payload to the subscribed systems.

6. Demonstration

In this chapter, the use of the artifact is demonstrated. This includes the fourth stage as mentioned in the DSRM by Peffers et al. (2007). First, the development of the Proof of Concept is introduced. Next, Proof of Concept is described with the help of the different components. Finally, the testing process of the Proof of Concept is given.

The chapter begins by introducing the development of the Proof of Concept. This section provides an overview of the initial stages of the Proof of Concept. It also discusses the resources and strategies used in the development process.

Following this, the chapter delves into a detailed description of the Proof of Concept. This involves a comprehensive exploration of its different components. Each component is examined in terms of its function and how it contributes to the overall system. This section aims to provide a clear understanding of the Proof of Concepts structure and operation.

Finally, the chapter concludes with an explanation of the testing process of the Proof of Concept. This includes the methodologies used for testing and the interpretation of test results. This section underscores the importance of thorough testing in ensuring the Proof of Concepts effectiveness and reliability.

6.1 Proof of Concept

6.1.1 Development of the Proof of Concept

Development of the Artifact

For developing the artifact, the Mendix Low-Code Platform has been used. The choice for this platform is supported by the fact that the company where this project is conducted also uses this Low-Code platform. Further, the conducting researcher has experience with Low-Code Application Development.

Development Methodology

The methodology for developing the Proof of Concept, is the Scrum methodology. Within this methodology, different pre-defined user stories will be planned in different phases called sprints. For this project, it was chosen to use sprints of one week. After each sprint, a retrospective will be held, which includes discussing the progress of the development and the planning of the next sprint.

6.1.2 Developed Proof of Concept

The Proof of Concept consists of four different components. The four components are described below. Screenshots of the Proof of Concept can be found in Appendix C at the end of this report.

Homey Pro

The first component is the Homey Pro, which acts as a gateway between the IoT devices and the MQTT broker. The Homey Pro simulates the different IoT devices from the different cases. This gateway is connected to the MQTT broker and publishes the messages to one of the pre-defined topics on the MQTT broker. For every case, a virtual device has been created in Homey, which publishes to the specified topic on the MQTT broker when activated.

MQTT Broker

The second component is the MQTT broker. This broker runs in the cloud and is supplied by Flespi.io, which is a free IoT communication platform. The Gateway produces the messages to a topic on this broker, after which the systems that subscribed to this topic receive these messages.

Prototype

The third component is the developed prototype in Mendix. This consists of an abstract iPaaS for integrating IoT devices with the help of Semantic Interoperability. The Mendix application runs on the localhost of the researcher. The prototype consists of five phases, all supporting the Integration Specialist while integrating IoT sources. These phases follow the phases as mentioned in the Solution Design. These phases were based on the Integration Lifecycle management used by the iPaaS as provided by the sponsor. These phases can be compared to the phases of the B-model as proposed by Birell & Ould (1988). A more detailed description of these phases is given in the design choices chapter of this thesis.

The first phase is the capture phase. This phase consists of creating the systems within the iPaaS environment. The source systems are the IoT gateways which need to be integrated, the target systems are the systems which need to receive the transformed messages. Within the edit page of a system, the details need to be specified. These details consist of the system name, technical name, message format, MQTT proxy, and MQTT topics.

The second phase is the design phase. This phase consists of designing the integrations with the help of the source messages, message mapping, and a global ontology. First the source schema needs to be extracted from the IoT gateway. This is done by subscribing to an MQTT topic as specified in the systems details page. When a message is received on that topic, the system extracts the schema of the incoming message with the help of parsing the JSON objects. The schema of the message is visualized in an entity and the attribute types are automatically extracted from the incoming message. The source schema and the attribute types can be edited if it does not completely correspond to the source schema definition.

After saving the source schema, a mapping can be made to the global ontology of the iPaaS environment. This global ontology consists of the core IoT ontology (ETSI – SAREF in this case), and the selected domain ontology extensions. These ontologies can be loaded by importing the JSON-LD files from the ontologies. This is done by parsing the JSON-LD files and saving the extracted classes and relationships in the database. The domain ontologies can be added or removed, while the core ontology cannot be deleted. The mapping between the source schema and the global ontology is done semi-automatically and can be edited if the mapping needs to be changed. This mapping is based on the attribute names and types of the incoming source schema.

The third phase is the create phase. In this phase the integration flow can be created in a low-code manner. To add the transformation into the flow, the Transform to Global Ontology event needs to be clicked. Next, a target system needs to be selected. These target systems have been initiated in the Capture phase. The flow represents the paths that the incoming message will take to arrive at the target system. The transformed message will be published to the Kafka topic which belongs to this target system.

The fourth phase is the deploy phase. In this phase the Semantic Adapter is generated. The Semantic Adapter is a Java-file which defines an input topic, an output topic, and a set of transformation rules. The transformations create a message with the necessary JSON-LD components. The values of these JSON-LD components are created by parsing the incoming JSON message and creating variables of the relevant attributes. By clicking on Deploy, the iPaaS environment downloads the Semantic Adapter on the client of the user. This file needs to be copied to the Kafka Stream Processing folder of the Kafka broker to run the application. In this Proof of Concept, Eclipse is used to run the generated Java application. The Java application processes the incoming JSON messages and returns a transformed message to the output topic.

The fifth phase is the manage phase. In this phase the integration can be tested. In the left panel, the source message of the IoT system is given. In the centre, the iPaaS is visually represented, and in the right panel, the output message can be seen. The user first needs to consume the output topic of the Target System, by clicking on the Consume topic button. After the topic is consumed by the iPaaS environment, a test message can be published to the Kafka Broker by clicking on Produce to Kafka. The Kafka Broker receives the message on the incoming topic and transforms it according to the Stream Processor/Semantic Adapter. The transformed message will be published to the topic of the target system. The right panel will show the transformed message as received on the output topic of the Kafka Broker in JSON-LD format.

Kafka Broker
The fourth component is the Kafka broker, which runs on the localhost of the researcher. This broker has been selected because it is one of the most popular solutions for Event Streaming within integration platforms (Chatt, 2019). The Kafka broker uses Stream Processors which consists of a set of rules which state how the incoming payload needs to be transformed. The Stream Processor in this case is a Java application. This application first consumes a Kafka topic, next transforms the message, and finally sends the transformed message to the outgoing topic.

6.1.3 Testing and Refinements

For the testing of the Proof of Concept, the Unit Test methodology is used. A unit test exercises a "unit" of code in isolation and compares the actual with the expected results (Olan, 2003). The unit test used in this research consists of objectives, steps, expected results, and actual results. As described in the solution design, the Proof of Concept is divided into five phases. Each of these phases contributes to the semantically enrichment of the IoT integrations. Below each phase is given with its test units.

1. Capture Phase
The Capture Phase consists of one unit which involves defining the source (IoT) and target systems. For the source IoT systems, the user should be able to define an MQTT Proxy, and a matching MQTT topic. The list of options for the MQTT Proxy is defined in the environment settings. All the input fields need to be filled to save a created system.

1.1 Test Case: System Creation:

- Objective: Verify that the systems can be created within the iPaaS environment.
- Steps:
0. Be sure that an MQTT Proxy is configured within the iPaaS environment.
1. Navigate to the system creation page.
2. Enter the required details for a new system.
3. Submit the form to create the system.
- Expected Result: A new system should be created within the iPaaS environment.

Test Result:
The test has the expected result. A system with the name *Test IoT Source* has been created. The technical name of this systems has been generated as *TstITSrc*, the message format has been set as JSON, a pre-configured MQTT proxy has been selected, and the relevant MQTT topic has been set as *weighbridge*.

The same test has been executed for target systems. In this case, a system with the name *Test Target System* has been created. This system got *TstTrgtSstm* as the generated technical name, the message format has been set as JSON, and the target topic has been set as *TargetTopicOut*.

2. Design Phase

The Design Phase consists of multiple units like extracting the source schema, mapping of the source schema to the global ontology, and defining the global ontology.

2.1 Test Case: Extract Source Schema:

- Objective: Verify that the integrations can be designed within the iPaaS environment using MQTT and source schema extraction.
- Steps:
0. Be sure that a source IoT system has been created with an input topic and an MQTT topic.
1. Navigate to the Design Phase page and select the systems details.
2. Subscribe to the specified MQTT topic to extract the source schema from the IoT gateway.
3. Wait for a message on the MQTT topic and parse the JSON objects of the incoming message.
4. Visualize the schema of the message in an entity and automatically extract the attribute types from the incoming message.
5. Edit the source schema and the attribute types if necessary.
6. Save the source schema.
- Expected Result: A correct schematic representation of the received IoT source message.

Test Result:
The test has the expected result. The source schema of the system *Test IoT Source* follows the schematic representation as intended by the source system.

2.2 Test Case: Import Domain Ontologies:

- Objective: Verify that the relevant domain ontologies can be imported within the iPaaS environment.
- Steps:
0. Be sure that a relevant core ontology is present within the iPaaS environment.
1. Navigate to the global ontology page.
2. Search a relevant domain ontology.
3. Import the selected domain ontology.
4. Visualize the imported domain ontology in triple format, with the mapping to the core ontology.
- Expected Result: A correct visual representation of the imported domain ontology, following the classes and relations of the JSON-LD specifications of the imported ontology.

Test Result:
The test has the expected result. The imported domain ontology, in this case *saref4auto*, follows the classes and relations of the JSON-LD specification as intended in the source of the domain ontology.

2.3 Test Case: Map Source Schema to the Global Ontology:

- Objective: Verify that the source schema can be correctly mapped to the global ontology of the environment.
- Steps:
0. Be sure that a source IoT system schema has been extracted using the MQTT proxy, and that a relevant and complete global ontology is imported.
1. Navigate to the schema mapping page.
2. Click on the Match Schema button.

3. Manually match the unmatched attributes with the relevant classes.
- Expected Result: A correct visual representation of the mapping between the extracted source schema and the global ontology of the environment.

Test Result:
The test has the expected result. The extracted source schema of the *Test IoT Source* has been matched correctly to the global ontology, and the relations and classes are correctly represented in a visual manner.

3. Create Phase
The Create Phase consists of the low-code integration flow visualiser. This involves selecting the transformations and the target system.

3.1 Test Case: Create an Integration Flow:
- Objective: Verify that integration flows can be created within the iPaaS environment.
- Steps:
0. Be sure that a source IoT schema system has been mapped to the global ontology, and that a target system is created with a pre-defined output topic.
1. Navigate to the Create Phase page and select the system details.
2. Click on the Transform to Global Ontology event.
3. Select a target system as the send target system.
- Expected Result: A correct visual representation of the integration flow between a source IoT system and a target system.

Test Result:
The test has the expected result. The integration flow correctly represents the integration as intended, showing the source system, transformations, and the target system.

4. Deploy Phase
The Deploy Phase consists of the semantic adapter generator. The semantic adapter is the source code of a Java application which can be run as a stream processor within a Kafka broker.

4.1 Test Case: Generate Semantic Adapter:
- Objective: Verify that a correct semantic adapter is being generated by the iPaaS environment.
- Steps:
0. Be sure to first start the Zookeeper and the Kafka Broker, a flow must be created with a source IoT system mapped to the global ontology, and a target system with a pre-defined output topic needs to be in place.
1. Navigate to the Deploy Phase page and select the system details.
2. Click on the Deploy button.
3. Copy the downloaded Java-file with the semantic_adapter_[technical name] as filename to the folder of the Kafka Stream Processor.
4. Run the Java-file as Java application.
- Expected Result: A syntactic correct generated Java-file containing no errors running on the Kafka Stream Processor.

Test Result:
The test has the expected result. The generated Java-file *semantic_adapter_TstITSrc* is valid according to the used Eclipse Java IDE. The input and output topics follow the specified topics and systems as mentioned in the Capture Phase, and the semantic adapter can be run as Kafka Stream Processor.

5. Manage Phase

The Manage Phase consists of an overview with the sample source schema in JSON and the final output schema in JSON-LD. This final output schema listens to the output schema The semantic adapter is the source code of a Java application which can be run as a stream processor within a Kafka broker.

5.1 Test Case: Test the Deployed Semantic Adapter:
- Objective: Verify that the deployed semantic adapter transforms the incoming source schema in a correct way.
- Steps:
0. Be sure to first start the Zookeeper, Kafka Broker, and the generated Semantic Adapter.
1. Navigate to the Manage Phase page and select the systems details.
2. Click on the Consume [topic] button to consume to the output topic of the running semantic adapter.
3. Click on the Produce message button to produce the sample payload to the input topic of the running semantic adapter.
4. Wait for the transformed payload to be shown.
5. Validate the transformed payload with an online tool like a JSON schema validator.
- Expected Result: A syntactic and semantic correct transformed JSON to JSON-LD message, valid according to a validator.

Test Result:
The test has the expected result. The generated Java-file *semantic_adapter_TstITSrc* transforms the sample source payload into valid JSON-LD according to the used JSON-LD validator (JSON for Linking Data, n.d.).

6.2 Conclusions

To conclude, the developed Proof of Concept has been demonstrated with the different developed components. The overall Proof of Concept consisted of a Homey Gateway, MQTT Broker, Prototype iPaaS, Kafka Broker, and a Kafka Stream Processor. The Proof of Concept has been tested following the Unit Test method, all these tests had the expected outcomes.

7. Evaluation

In this chapter, the treatment is evaluated. This is part of the fifth stage as mentioned in the DSRM by Peffers et al. (2007). This chapter seeks the answer to the fourth research question stated below.

(SQ4) - What can be the impact of iPaaS architectures that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems based on three specific cases?

First, the designed treatment will be validated with the help of the potential impacts, design effectiveness, and the sensitivity of the designed treatment.

Next, the same points will be validated based on the results of the proof of concept. This was done by presenting the treatment in the form of a Proof of Concept to the stakeholders. This involved the expert opinion on the impacts, design effectiveness, and the sensitivity of the designed treatment with the stakeholders.

Finally, the treatment will be validated with the help of three conducted case studies. These case studies consisted of relevant cases and helped to validate the treatment with the help of a tangible Proof of Concept.

7.1 Design Validation

7.1.1 Design validation method

The methodology used for design validation, as proposed by Wieringa & Morali (2012), is an approach that ensures the effectiveness and relevance of a design in given problem context. This methodology consists of four steps which are stated below.

The first step is an analysis of the design's potential impacts. It formulates the expected effects to ensure the design aligns with its intended outcomes. Next, the design is assessed against the stated criteria, evaluating its effectiveness. This step measures the design's success by comparing expected and actual effects. The third step compares the design's performance with other solutions, ensuring it is the most effective and efficient. The final step examines the design's sensitivity to problem changes, understanding how these might affect its effectiveness. This ensures the design remains relevant as the problem evolves.

This methodology validates and continuously improves the design. It provides a framework for regular assessment and adjustment, improving the design's adaptability and durability.

7.1.2 Results based on the Solution Design

The design has been validated with the help of the relevant stakeholders within this project. The different designs have been presented and accorded. This was needed to get a *Go* for the development of the prototype. For the validation of the design, the Design Validation of Wieringa & Morali (2012) is used.

Potential Impacts

The potential impacts of the design are based on the stated conceptual framework as mentioned in this research. The conceptual framework stated that Semantic Interoperability positively impacts the development effectiveness, governance, and the seamlessness of IoT integrations on iPaaS.

Semantic Interoperability enhances development effectiveness by establishing a common language for data interpretation, reducing errors, and speeding up the development process. This leads to cost savings and improved project timelines. Further, Semantic Interoperability is crucial for governance, enabling effective communication and data exchange for decision-making processes. This results in

improved operational efficiency, risk management, and compliance. Finally, in the IoT, Semantic Interoperability ensures that the diverse devices and platforms can interact seamlessly on iPaaS. This enhances user experience, improves system performance, and enables new functionalities.

Design Effectiveness
The designs are based on the requirements of the stakeholder and on comprehensive analysis of how Semantic Interoperability for IoT integrations can be incorporated in iPaaS architectures. There are different criteria stated in this project to which the treatment must comply. The requirements of the stakeholders have been considered in designing the treatment. Not all requirements could be satisfied by the treatment, but the essential requirements as negotiated with relevant stakeholders have been met.

The requirements which have not been satisfied consist of functionalities which do not directly add to the value of the research itself but could be a nice feature for a final solution.

The architecture criteria as identified in the research also have been satisfied. These architectural criteria formed the basis of the design architecture. When comparing the results of the research with the designed architecture, the core components as identified in the research can be found back.

Comparison with other Treatments
The report compares the proposed treatment with existing ones, highlighting the unique capability of the former to enable Semantic Interoperability for IoT integrations on iPaaS. While some iPaaS solutions support IoT integrations, they lack this feature. Semantic Interoperability, the ability for systems to exchange data with shared meaning, is crucial for the different IoT devices and applications to communicate seamlessly.

The proposed treatment stands out for its focus on Semantic Interoperability within the context of IoT integrations on iPaaS architectures, unlike other treatments that concentrate more on the analysis of linked data or linked open data. These treatments, while insightful, may not directly tackle the challenges of Semantic Interoperability in the same context. Therefore, the proposed treatment adds value to the existing treatments, collectively advancing Semantic Interoperability solutions.

Design Sensitivity
The proposed solution in this research is designed to address the persistent issue of IoT device heterogeneity. The diverse data notations, protocols, and specifications of IoT devices from various manufacturers pose a significant interoperability challenge.

While an ideal scenario would have all devices designed with Semantic Interoperability in mind, the reality is different. Existing devices often do not align with this concept, making it complex to implement from the start.

The Semantic Adapter discussed in this research addresses this problem by enabling Semantic Interoperability among the different IoT devices using iPaaS. As companies continue to use IoT integrations due to their potential, the Semantic Adapter meets an ongoing need in the IoT world, emphasizing the importance of this research.

7.1.3 Results based on Proof of Concept
The treatment has been further validated in the form of a Proof of Concept with the relevant stakeholders within this project. The treatment has been presented and accorded. This was needed to make sure the prototype was in accordance with the stakeholders. To validate with the help of the Proof of Concept, it has been chosen to use the Design Validation of Wieringa & Morali (2012). This method is also used to validate the design, but this time, only the Impacts and Effectiveness will be

validated based on the Proof of Concept. It has been chosen to not include the Comparison with other treatments and the Sensitivity of the design, because these will have the same outcomes as in the previous chapter.

Impacts
The impacts of the treatment are based on the stated potentials in the conceptual framework as stated in this research. The conceptual framework stated that Semantic Interoperability positively impacts the development effectiveness, governance, and the seamlessness of IoT integrations on iPaaS. The impacts have been discussed with the relevant stakeholders with the Proof of Concepts as a tangible representation of the treatment.

The developed Proof of Concepts, based on the treatment, positively impacts the development effectiveness, governance, and seamlessness of IoT integrations on iPaaS. The Semantic Interoperability enhances the development effectiveness by providing a common data representation, reducing errors, and speeding up the process. Intelligent mapping could further improve effectiveness. Semantic Interoperability further improves the governance by providing a comprehensive view of the data flow, leading to informed decisions. A dashboard could provide even more insights. Finally, the treatment enables seamless integration of various IoT devices within an iPaaS environment. Implementing different ontologies into the platform's global ontology could further enhance seamlessness.

Looking at the stated objectives and success criteria, it can be stated that they all have been met with outcomes of the project.

Effectiveness
The effectiveness of the treatment according to the stakeholders can be analysed based on the stated goals of these stakeholders. These goals have been identified in the objectives chapter of this research. The stakeholders which will be used for this validation are the Integration Specialists, and the Product Owners/Managers. The Proof of Concept has been used as tangible representation of the treatment and has been discussed with these stakeholders.

Integration Specialists
Their goal is to integrate IoT sources more efficiently and effectively. This is achieved using the MQTT Proxy and the global ontology, which allow rapid extraction of the source schema and easy mapping to a standard model. The effectiveness of the IoT integrations is ensured by transforming the source schema to a standard representation, achieving Semantic Interoperability. The treatment has been further evaluated with this stakeholder. The interviewed Integration Specialist acknowledged that the requirements have been satisfied. Further, the Integration Specialist acknowledged the improved development efficiency and governance provided by the treatment.

Product Owners/Managers
Their goal is to maximize product value and represent the voice of the customer. The product value is increased by incorporating the semantic adapter, enabling efficient and effective integration of IoT sources into their iPaaS environment. This also addresses various customer requests. The treatment has been further evaluated with this stakeholder. The Product Manager acknowledged that the requirements have been satisfied.

7.2 Case study
To validate the proposed solution, three cases have been selected. These cases relate to relevant request that the company has received. The cases studies will use a Homey Pro as gateway, which

simulates the messages from the different devices. The gateway is connected to the Proof of Concept using an MQTT broker.

The method used for this case study follows an exemplification of the theory as constructed in the theoretical framework. The purpose of these case studies is to demonstrate the constructed artifact.

7.2.1 Case 1: Wearable Scanner

The first selected case is a real-world case the company executed. In this case, a postal company uses a ProGlove for scanning parcels in their warehouse.

The ProGlove is an innovative solution designed to enhance operational efficiency and reduce physical strain in logistics and transportation settings. This smart wearable integrates a scanner into a glove, providing workers with a hands-free solution for quick and accurate scanning of goods.

The wearable will be connected to a local gateway, which is integrated to the iPaaS with the help of the MQTT protocol. This integration ensures the communication between the ProGlove and the company's ERP system. The iPaaS serves as a mediator, managing the flow of data between the wearable device and the target system.

A real-world message of the ProGlove will be used to demonstrate this case. This message has been retrieved from an Integration Specialist working on that project and will be simulated by the Gateway.

7.2.1.1 Case Set-up

To execute the case, first a virtual device must be created in the gateway. This device will be called ProGlove. When the ProGlove is activated, a message will be published to the *'logistics/proglove/scannedData'* topic of the MQTT broker. The payload of this message can be seen in figure 55 and is based on the real payload used by the ProGlove.


```
{
  "api_version":"1.0",
  "event_type":"scan",
  "event_id":"02114-ae-46e3-8b00-a3f7ea8672dg",
  "time_created":1694419717125,
  "scan_code":123456789012,
  "device_serial":"MDM425447",
  "gateway_serial":"PGGWWW110109"
}
```


Figure 55: Used payload of the ProGlove case

This message needs to be mapped to the global ontology which, in this case, consists of the ETSI SAREF Core, the SAREF extension for the industry and manufacturing domain (SAREF4INMA), and the SAREF extension for Wearables (SAREF4WEAR). The final mapping between the source schema and the global ontology can be seen in figure 56. The values after the colon represent the attribute names of the source schema, while the value in the brackets refer to the values of the sample message.

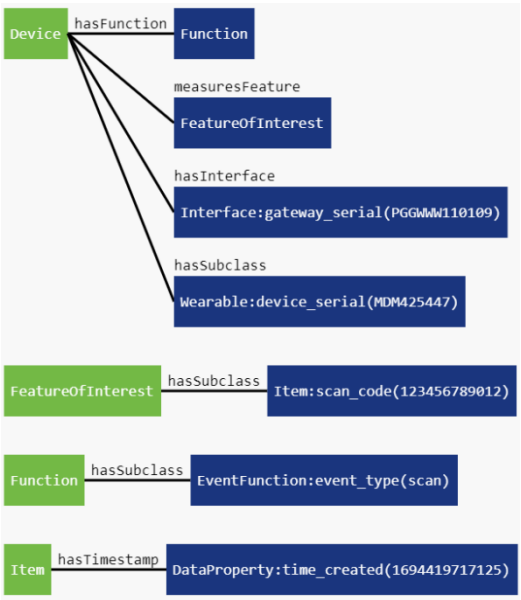

Figure 56: Mapping between source schema and global ontology for the ProGlove case

With the mapping between the source schema and the global ontology, the semantic adapter can be generated. This is a Java application which can be used in the Stream Processor. Each semantic adapter is unique, as it contains the transformation rules for the incoming payload of one specific IoT integration. The semantic adapter also includes the input and output topics as given while creating the systems in the capture phase. The transformed payload which has been sent to the output topic can be seen in figure 57.


```
{
   "@context":{
      "owl":"http://www.w3.org/2002/07/owl#",
      "s4wear":"https://saref.etsi.org/saref4wear/",
      "saref":"https://saref.etsi.org/core/",
      "s4inma":"https://saref.etsi.org/saref4inma/"
   },
   "@type":"s4wear:Wearable",
   "owl:hasValue":"MDM425447",
   "s4wear:hasInterface":{
      "@type":"s4wear:Interface",
      "owl:hasValue":"PGGWWW110109"
   },
   "saref:hasFunction":{
      "@type":"saref:EventFunction",
      "owl:hasValue":"scan"
   },
   "s4wear:measuresFeature":{
      "@type":"s4inma:Item",
      "owl:hasValue":"123456789012",
      "saref:hasTimestamp":"1694419717125"
   }
}
```


Figure 57: Transformed payload by the Semantic Adapter for the ProGlove case

7.2.1.2 Case Results

The results are as expected. With the help of the prototype, the ProGlove message has been translated into valid JSON-LD, which has been published on the Kafka Broker.

The message as provided by the integration specialist was abstract, having no clear description of the context. Because of the abstractness of the attribute names, automatic mapping to the ontology was difficult, but with the help of the search function in the Proof of Concept, the relevant classes could be found for this case. This led to an efficient integration process, which could be even more efficient if a better, more intelligent matching algorithm has been used.

7.2.2 Case 2: Weighbridge

The second selected case is based on a request the company got but has not been fulfilled. This case focuses on the integration of a weighbridge system and an ERP system of a company. Currently, the weight data from trucks is manually entered into the ERP system through a display. The goal is to automate this process for increased accuracy and efficiency.

The weighbridge system will be equipped with an IoT sensor to automatically capture the weight data. This data will be transmitted to the ERP system, eliminating the need for manual data entry. The integration will be facilitated by the iPaaS, which communicates between the weighbridge and the ERP system.

The message of the Weighbridge has been established by the researcher and will be simulated by the Gateway.

7.2.2.1 Case Set-up

To execute the case, first a virtual device must be created in the gateway. This device will be called Weighbridge. When the Weighbridge is activated, a message will be published to the *'weighbridge/truck_weight'* topic of the MQTT broker. The payload of this message can be seen in figure 58.


```
{
   "timestamp":"2023-11-27T13:20:10Z",
   "device_id":"weighbridge_001",
   "truck_id":"truck_123",
   "weight":15000,
   "unit":"kg"
}
```


Figure 58: Used payload of the Weighbridge case

This message needs to be mapped to the global ontology which, in this case, consists of the ETSI SAREF Core, the SAREF extension for the agriculture and food domain (SAREF4AGRI), the SAREF extension for automotive (SAREF4AUTO), and the SAREF extension for Wearables (SAREF4WEAR). The final mapping between the source schema and the global ontology can be seen in figure 59. The values after the colon represent the attribute names of the source schema, while the value in the brackets refer to the values of the sample message.

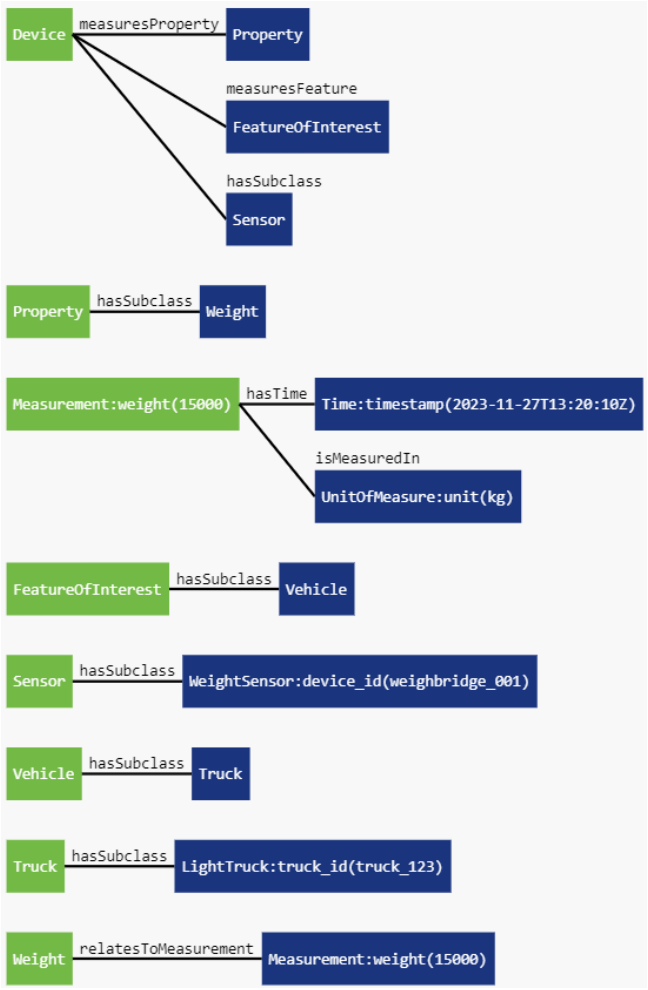

Figure 59: Mapping between source schema and global ontology for the Weighbridge case

With the mapping between the source schema and the global ontology, the semantic adapter can be generated. The transformed payload which has been sent to the output topic can be seen in figure 60.

```
{
  "@context": {
    "owl":"http://www.w3.org/2002/07/owl#",
    "s4auto":"https://saref.etsi.org/saref4auto/",
    "s4agri":"https://saref.etsi.org/saref4agri/",
    "saref":"https://saref.etsi.org/core/",
    "s4wear":"https://saref.etsi.org/saref4wear/"
  },
  "@type":"s4agri:WeightSensor",
  "owl:hasValue":"weighbridge_001",
  "saref:measuresProperty": {
    "@type":"s4auto:Weight",
    "saref:relatesToMeasurement": {
      "@type":"saref:Measurement",
      "owl:hasValue":"15000",
      "saref:hasTime": {
        "@type":"saref:Time",
        "owl:hasValue":"2023-11-27T13:20:10Z"
      },
      "saref:isMeasuredIn": {
        "@type":"saref:UnitOfMeasure",
        "owl:hasValue":"kg"
      },
      "s4wear:measuresFeature": {
        "@type":"s4auto:LightTruck",
        "owl:hasValue":"truck_123"
      }
    }
  }
}
```


Figure 60: Transformed payload by the Semantic Adapter for the Weighbridge case

7.2.2.2 Case Results

The results are as expected. With the help of the prototype, the Weighbridge message has been translated into valid JSON-LD, which has been published on the Kafka Broker.

The message as used in this case was less abstract compared to the first case, having a clearer description of the context. This resulted in a better automatic mapping between the source schema and the global ontology. Only the truck_id and device_id have not been mapped to the global ontology. This could be improved by incorporating a more intelligent algorithm for mapping these attributes. In this case, the mapping of these remaining attributes has been made by using the search functions for the classes.

7.2.3 Case 3: Carpet equipped with a RFID-tag

The third selected case is based on a request the company got but has not been fulfilled. This case involves the use of RFID technologies to track and manage the inventory of a textile cleaning company. A company offering a service for maintaining and cleaning carpets equipped these carpets with cost-effective RFID tags. This enables real-time monitoring of their physical locations and updating inventory records accordingly.

When an employee of the company replaces a carpet at a client, the RFID-tag of the carpet is read by an RFID-reader in the vehicle of the employee. When the tag is read, the location of the carpet will be retrieved. The RFID reader is connected to a gateway which sends the data to the iPaaS, after which the location of the carpet is updated in the inventory management system.

The message of the Weighbridge has been established by the researcher and will be simulated by the Gateway.

7.2.3.1 Case Set-up

To execute the case, first a virtual device must be created in the gateway. This device will be called Carpet. When the Carpet is activated, a message will be published to the *'inventory/textile_location'* topic of the MQTT broker. The payload of this message can be seen in figure 61.


```
{
   "readerId":"Reader123",
   "time":"2023-11-27T13:20:10Z",
   "tagId":"RFID123",
   "location":{
      "long":12.34,
      "lat":56.78
   }
}
```


Figure 61: Used payload of the Carpet case

This message needs to be mapped to the global ontology which, in this case, consists of the ETSI SAREF Core, the SAREF extension for the industry and manufacturing domain (SAREF4INMA), and the SAREF extension for Wearables (SAREF4WEAR). The final mapping between the source schema and the global ontology can be seen in figure 62. The values after the colon represent the attribute names of the source schema, while the value in the brackets refer to the values of the sample message.

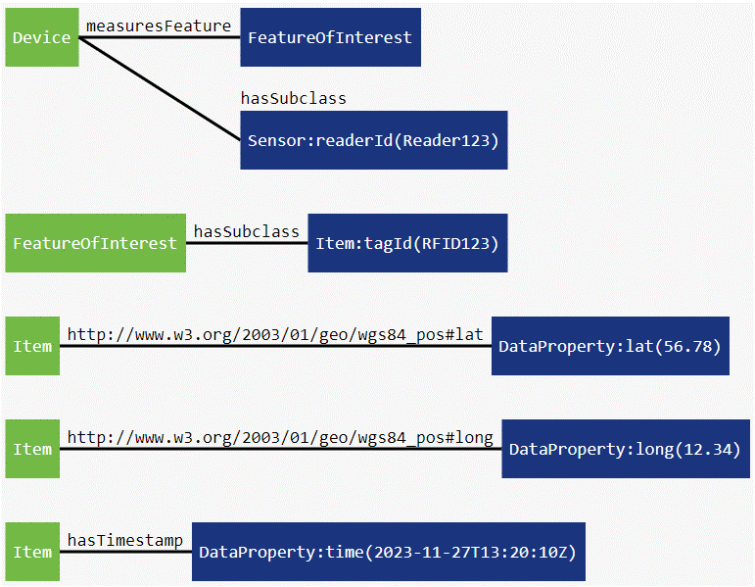

Figure 62: Mapping between source schema and global ontology for the Carpet case

With the mapping between the source schema and the global ontology, the semantic adapter can be generated. This is a Java application which can be used in the Stream Processor. Each semantic adapter is unique, as it contains the transformation rules for the incoming payload of one specific IoT integration. The semantic adapter also includes the input and output topics as given while creating the systems in the capture phase. The transformed payload which has been sent to the output topic can be seen in figure 63.

```
{
   "@context":{
      "owl":"http://www.w3.org/2002/07/owl#",
      "s4inma":"https://saref.etsi.org/saref4inma/",
      "saref":"https://saref.etsi.org/core/",
      "s4wear":"https://saref.etsi.org/saref4wear/"
   },
   "@type":"saref:Sensor",
   "owl:hasValue":"Reader123",
   "s4wear:measuresFeature":{
      "@type":"s4inma:Item",
      "owl:hasValue":"RFID123",
      "http://www.w3.org/2003/01/geo/wgs84_pos#long"
      :"12.34",
      "http://www.w3.org/2003/01/geo/wgs84_pos#lat":
      "56.78",
      "saref:hasTimestamp":"2023-11-27T13:20:10Z"
   }
}
```


Figure 63: Transformed payload by the Semantic Adapter for the Carpet case

7.2.3.2 Case Results

The results are as expected. With the help of the prototype, the Carpet message has been translated into valid JSON-LD, which has been published on the Kafka Broker. The used message was less abstract compared to the first case, but more abstract than the second case. This resulted in a challenging automatic mapping between the source schema and the global ontology. With the help of tweaking the algorithm, the challenge was able to be solved. For a next project this could be improved by incorporating a more intelligent algorithm.

7.3 Conclusions

The treatment has been validated using two different methods. First, the design has been validated following the Design Validation of Wieringa & Morali (2012). The potential impact of the designed treatment aligns with the proposed conceptual framework. The designs effectiveness is consistent with the objectives of the stakeholders. The treatment stands out from others by facilitating Semantic Interoperability for IoT integrations on iPaaS.

Other treatments either support IoT integrations without Semantic Interoperability or concentrate more on data interlinking for analytical purposes, rather than focusing on enterprise integrations. This unique focus sets the treatment apart in these fields.

The designed treatment does not seem sensitive to changes in the problem. The amount of IoT devices keep increasing, while these devices keep their heterogeneous nature.

To get a better understanding of the treatment, the Proof of Concept has also been validated with the stakeholders. To validate with the help of the Proof of Concept, it has been chosen to use the Design Validation of Wieringa & Morali (2012). This method is also used to validate the design, but this time, only the Impacts and Effectiveness will be validated based on the Proof of Concept. It has been chosen to not include the Comparison with other treatments and the Sensitivity of the design, because these will have the same outcomes as in the previous chapter. The impacts of the treatment based on the Proof of Concept follow the stated goals of the stakeholders. The effects of the treatment based on the Proof of Concep follow the stated conceptual framework.

The treatment has also been validated with the help of three case studies. The Proof of Concept served as a practical platform for evaluation. All results were aligned with the expectations, concluding that Semantic Interoperability plays an important role in the ProGlove, Weighbridge, and Carpet cases. It improves integration efficiency, governance of integrations, and seamlessness of these integrations. The research question which is answered by these cases is stated below.

(SQ4) - What can be the impact of iPaaS architectures that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems based on three specific cases?

The conceptual framework state that Semantic Interoperability has a positive effect on the Efficiency, Governance, and Seamlessness of IoT integrations. For each independent variable, a substantiation is given.

The use of different ontologies improved the efficiency of creating the integration in all three cases. However, the level of abstractness in the messages varied, affecting the automatic mapping to the ontology. The ProGlove case was the most abstract, making automatic mapping difficult, while the Weighbridge case was less abstract, resulting in better automatic mapping. The Carpet case, with its more complex payload using nested JSON objects, presented a challenging automatic mapping scenario. In all cases, the efficiency could be further improved with a more intelligent mapping algorithm. Further, the accuracy of a mapping could be added to show the confidence of a mapping between a source schema attribute and a global ontology class.

The governance of the integration was improved in the cases by providing a clear overview of the source systems, transformations, and target systems. The low-code aspect of the iPaaS and the semantic representation of the data further improved the governance. An overview of running integrations by incorporating a dashboard showing relevant metrics, could be added to provide even better governance.

The ProGlove case proved that it is possible to transform a real case into a semantically interoperable IoT integration. This ensures the seamless integration of IoT sources by first transforming the messages into a widely used standard notation (ontology), which can then be used by other systems using these notations. The seamless integration could be further improved with a SPARQL endpoint, ensuring that systems using SPARQL queries can easily retrieve the data used on the platform.

It must be stated that the cases have been established in accordance with the sponsor. This resulted in three relevant cases. But for each case, one example message has been used. It was chosen to create each message with a different structure, but more example messages and cases could even further validate the treatment.

Overall, the treatment has been validated with the help of the Design Validation of Wieringa & Morali (2012), both as design, and as Proof of Concept. Further validation has been conducted with the help of three case studies. It can be concluded that Semantic Interoperability can be seen as a powerful concept for improving the integration processes, and with further enhancements, it can provide even more benefits.

8. Conclusion and Discussion

This chapter will delve into a comprehensive discussion of the conclusions and results, guided by the conclusions, contributions, limitations, and subsequent recommendations.

8.1 Conclusion

This research aimed to identify how Semantic Interoperability could enhance IoT data integrations on iPaaS architectures for client organizations. The first research question which has been addressed in this thesis was “How can IoT integrations be incorporated on iPaaS architectures?”. This question resulted in a general architecture for implementing IoT in iPaaS architectures as can be seen in figure 64. Different IoT protocols that could enable these integrations on iPaaS architectures have been identified, with each protocol having different implementation solutions.

Based on the research, the MQTT protocol has been chosen as a suitable solution for IoT integrations on iPaaS architectures. The MQTT Proxy has been identified as an enabler for connecting IoT Gateways to iPaaS architectures. The iPaaS architecture itself is built upon the Kafka framework. In the research, the Kafka framework has been identified as a broadly used solution for integration platforms. Kafka follows the Event Streaming pattern, the same pattern which is used in the IoT.

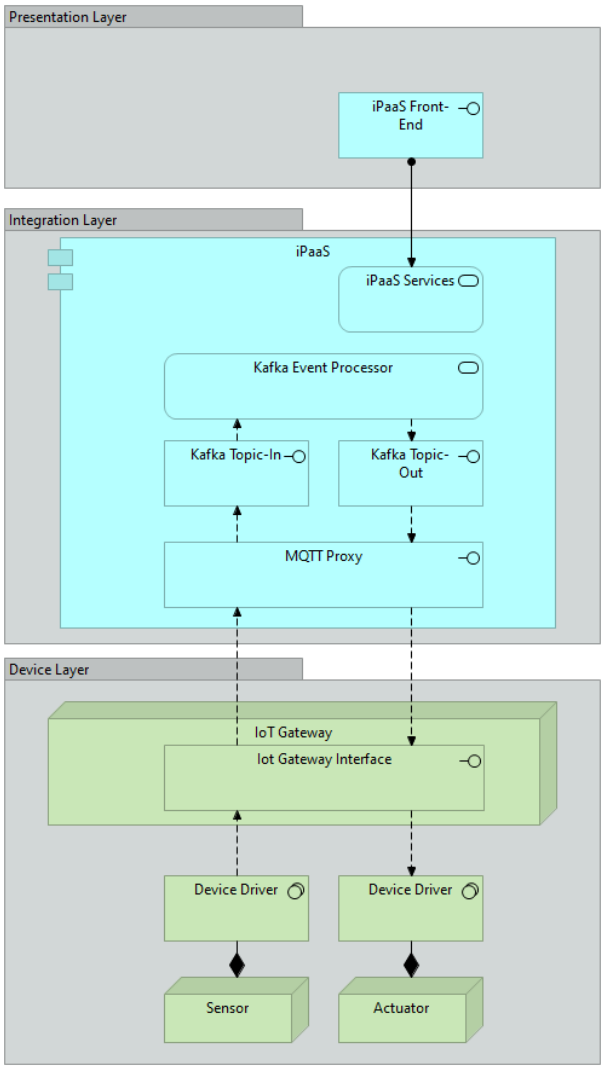

Figure 64: Combined IoT and iPaaS Architecture

The second research question which has been addressed in this thesis was “How can Semantic Interoperability be incorporated in iPaaS architectures?”. This question resulted in a general architecture for semantic interoperability in an iPaaS architecture as can be seen in figure 65. This architecture is based on different identified solutions for implementing Semantic Interoperability.

This architecture combines several existing treatments and concepts, such as the Semantic Adapter by Zehnder et al. (2020), Semantic Matching by Nie et al. (2021), and the IoT Platform Semantic Mediator concept by Ganzha et al. (2018). The Integration Manager by Jovanovic et al. (2021) is further used for generating different Semantic Adapter configurations. This automation simplifies the process of setting up and managing IoT integrations on the iPaaS platform, making it more user-friendly and efficient.

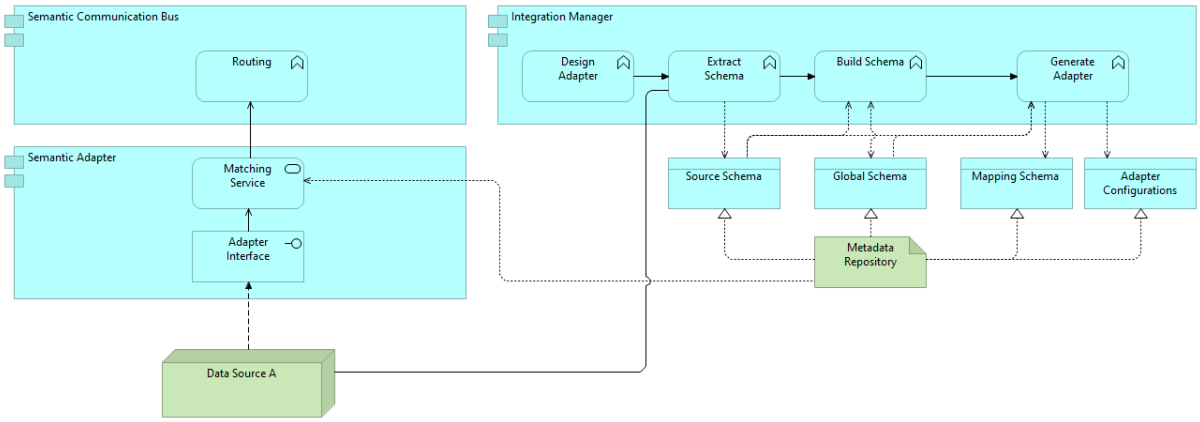

Figure 65: Semantic Interoperability Architecture

Based on the qualitative analysis on how IoT and Semantic Interoperability could be enabled on iPaaS architectures, a treatment has been designed. This treatment solves the heterogeneity problem of IoT integrations on iPaaS. Further, the designed treatment is also the answer to the third research question, which was “What are the key characteristics of an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across different systems?”. This treatment consists of five main components which have been identified in the qualitative analysis.

The first component is the Source Schema Extractor, which extracts the source schema from the given source IoT systems. Next, the Ontology-Schema Matcher matches the extracted source schema to a given Global Ontology within the platform. This Global Ontology is the next component and consists of a Core IoT Ontology and different Domain Specific Ontologies. Extending this Global Ontology with these Domain Specific Ontologies can be done by the next component, the Ontology Searcher & Matcher. The final component is the Semantic Adapter Generator. This component generates the configurations for the IoT Integrations in the form of a Java application which can be consumed by the running iPaaS architecture.

Based on the designed treatment, a Proof of Concept has been developed with the help of the Mendix Low-Code Platform. This Proof of Concept demonstrated that the designed treatment is technically feasible.

With the help of this Proof of Concept, the treatment could be evaluated based on three relevant IoT cases. These three cases help answer the fourth research question. This research question, SQ4, was “What can be the impact of iPaaS architectures that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems based on three specific cases?”. The answer to this question is

that the cases proved that Semantic Interoperability in fact can enhance the Development Effectiveness, Governance, and Seamlessness of IoT integrations on iPaaS architectures.

The effect of Semantic Interoperability on the Development Effectiveness of IoT data integrations on iPaaS architectures is positive. With the help of the semantic representations of the IoT data with the help of a global ontology, the time it takes to develop an IoT integrations in an iPaaS environment is reduced. This effect is because the Integration Specialists gets to understand the data better in a standard schematic and semantic manner. This removes the step of modelling a common data model from scratch, because a standard global ontology is in place.

The effect of Semantic Interoperability on the Governance in IoT data integrations on iPaaS architectures is also positive. This is achieved by providing a clear overview of the source systems, data transformations, and target systems. The low-code aspect of the iPaaS and the semantic representation of the data further improved the governance.

Finally, the effect of Semantic Interoperability on the Seamlessness of IoT data integrations on iPaaS architectures is also positive. The real-world case as used in the demonstration ensures the seamless integration of IoT sources by first transforming the messages into a widely used standard notation (ontology), which can then be used by other systems in the iPaaS environment. The standard notation ensured that systems could be seamlessly integrated together by transforming them into these standard notations.

The overall research question was, “How to design an iPaaS architecture that leverages Semantic Interoperability to seamlessly integrate IoT data across the systems of client organizations, to enhance development efficiency and governance in IoT application integrations?”. The answer to this question is given by the different sub research questions. To enable an iPaaS architecture to incorporate IoT, a fitting protocol needs to be selected. This research suggests incorporating the MQTT protocol, which is widely used in the IoT.

To leverage Semantic Interoperability for IoT within the iPaaS architecture, the Semantic Adapter in combination with a Global Ontology could be used. The designed treatment implements these approaches into an iPaaS architecture, which has been validated by executing three relevant cases on a Proof of Concept.

8.2 Contributions

Bispo (2023) argues that research contributions can be classified into different types, such as practical and theoretical. The practical contribution of this study consists of a proposed design which enhances the integration capabilities of iPaaS architectures by using Semantic Interoperability. The practicality of the design is further demonstrated by the development of a Proof of Concept. This Proof of Concept enables Integration Specialists to retrieve source schemas from IoT sources, efficiently matching these schemas to a global representation of data, and to deploy semantically enriched IoT integrations.

The theoretical contribution of this study consists of a proposed treatment which enables iPaaS architectures to semantically interoperate the heterogenous IoT appliances. This treatment combines different found architectures in the field of iPaaS, IoT, and Semantic Interoperability.

By extending an iPaaS platform architecture with Semantic Interoperability, the treatment contributes to the field of iPaaS by presenting a new approach to integrating IoT solutions on iPaaS. The current literature in the field of iPaaS does not focus on the Semantic Interoperability aspects of integrating with iPaaS, but more on utilizing iPaaS solutions in different contexts.

The theoretical contributions in the field of IoT consist of enabling a new approach for integrating devices into enterprise applications. The current literature in the field of IoT focusses more on generally describing the IoT in different appliances with its challenges.

For Semantic Interoperability, the theoretical contributions can be seen as providing a combination of existing treatments to enable the concept of Semantics on iPaaS architectures. The treatment, with respect to Semantic Interoperability, combines treatments such as the Semantic Adapter by Zehnder et al. (2020), Semantic Matching by Nie et al. (2021), and the IPSM concept by Ganzha et al. (2018). This combination of established treatments enables the creation of Semantically Interoperated IoT integrations on iPaaS architectures.

The Integration Manager by Jovanovic et al. (2021) is further used for generating different Semantic Adapter configurations. This automation simplifies the process of setting up and managing IoT integrations on the iPaaS platform, making it more user-friendly and efficient. These components help in understanding and translating the data schemas of different IoT sources, thereby enabling meaningful communication and data exchange among them.

8.3 Limitations

Proof of Concept

In order to realize a Proof of Concept within the period of this project, a few limitations have occurred. The first limitation is that the Proof of Concept only incorporates JSON, and JSON-LD as message structures within the platform.

Next, the Proof of Concept did not incorporate real target systems. The integration with the target systems have been simulated by subscribing to the topics of the Kafka broker with the same prototype. This was to test if the transformed messages were correct and were able to be received by a target system.

Finally, only extensions from the SAREF ontology have been used. This could be a limitation because there are many more ontologies in place. This does not impact the results of the study, because the purpose was to prove the effect of Semantic Interoperability on IoT integrations in iPaaS architectures.

iPaaS Research

The limited research in the field of iPaaS can be seen as a limitation of this study. While this research has made steps in understanding the role of Semantic Interoperability in iPaaS architectures, the limited existing research in this domain of iPaaS presents a considerable constraint. Future research should aim to build upon this study, contributing to a more comprehensive understanding of iPaaS architectures and their potential.

Validation

The treatment has been validated with three relevant cases. The first message is a real-world example as provided by an integration specialist. The other two messages were established by the researcher self. These messages were established to all differ from each other's structure, but even more example messages or cases could further validate the treatment.

8.4 Recommendations

Further, this research focussed on the concept of transforming incoming IoT messages from JSON format into JSON-LD format. Future research could include other relevant message formats like XML or RDF.

Mapping Algorithm
For the mapping between the source schema and the global ontology, a static mapping algorithm has been used. This algorithm made the matches based on the attribute names and types. A more intelligent matching algorithm could provide a better mapping solution.

SPARQL Endpoint
To make the artifact even more fitting for the semantic web, a SPARQL endpoint could be exposed by the platform. This endpoint enables sources on the web to consume the semantically enriched data with the help of SPARQL queries. With this, a query could be conducted which retrieves all devices which currently have a temperature higher than 30 degrees Celsius. This could make the solution even more useful.

Further Investigation of iPaaS
Further investigation into the domain of iPaaS is recommended. As mentioned in the limitations, the field of iPaaS research is limited to a few reliable studies. To improve the state of iPaaS research, different studies need to be conducted to get more reliable results in future studies. Given the interdisciplinary nature of iPaaS and IoT, research collaborations between computer scientists, engineers, data scientists, and industry practitioners could result in innovative insights and solutions for the given fields.

Further Investigation of Semantic Interoperability in iPaaS
While the current research has primarily focused on the integration of IoT sources, the scope of iPaaS architectures extends far beyond this. It is recommended to further research the aspect of Semantic Interoperability in iPaaS environments by investigating other application and integration types. Currently, a common data model is in place for integrating these appliances, but the effect of Semantic Interoperability on these integrations should be further researched. This could also improve the development efficiency, governance, and seamlessness of these integrations on iPaaS architectures. Semantic Interoperability could utilize the functionality of a common data model in the form of an ontology.

Further Validation of the treatment
As mentioned in the limitations, the treatment could be validated with more example messages or cases. Only three cases have been used, with all having one example message each. When more example messages will be validated with the help of the Proof of Concept, a better insight in the validity of the treatment could be given.

9. References

Abeysinghe. (2021). *The Role of EiPaaS in Enterprise Architecture*. Retrieved from WSO2: https://wso2.com/choreo/resources/the-role-of-eipaas-in-enterprise-architecture-part-2/

Allen & Garlan. (1997, July). A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*.

Altexsoft. (2019, May 16). *The Most Popular Prioritization Techniques and Methods*. Retrieved from Altexsoft Software R&D Engineering: https://www.altexsoft.com/blog/business/most-popular-prioritization-techniques-and-methods-moscow-rice-kano-model-walking-skeleton-and-others/

Ankulov. (2020, June 2). *Product owner vs. product manager: Who runs the show?* Retrieved from Productboard: https://www.productboard.com/blog/product-owner-vs-product-manager/

Aros & Torres. (2018). Implementing a Signing Forms mechanism in an open XMPP Server to reduce successful network attacks. *Conference: SSN 2018 IV School on Systems and Networks.*

Asemani et al. (2019). A Comprehensive Fog-Enabled Architecture for IoT Platforms. *Communications in Computer and Information Science*.

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management* , pp. 337-342.

Balakrishna & Thirumaran. (2019). Towards an Optimized Semantic Interoperability Framework for IoT-Based Smart Home Applications.

Bernardi et al. (2002). From UML Sequence Diagrams and Statecharts to analysable Petri Net models. *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*.

Berners-Lee et al. (2001, May). The Semantic Web. *Scientific American*.

Bernstein et al. (2021). *MQTT (MQ Telemetry Transport)*. Retrieved from TechTarget.

Betz et al. (2023). How Fast is My Software? Latency Evaluation for a ROS 2 Autonomous Driving Software. *IEEE Intelligent Vehicles Symposium, Proceedings*.

Bhatt et al. (2022). Implementing MQTT protocol IoT based for AMI Network of Smart Grid system. *Aut Aut Research Journal*.

Bigio, S. (2023). *How to Utilize a System Architecture Diagram*. Retrieved from BairesDevBlog: https://www.bairesdev.com/blog/how-to-utilize-system-architecture-diagram/

Birrell & Ould. (1988). A practical handbook for software development.

Bispo, M. d. (2023). Scientific Articles' Theoretical, Practical, Methodological, and Didactic Contributions.

Bonte et al. (2017). The MASSIF platform: a modular and semantic platform for the development of flexible IoT services. *Knowledge and Information Systems*.

Booch et al. (1996). The Unified Modeling Language for Object-Oriented Development. *Rational Software Corporation*.

Bourne & Kasperczyk. (2009). Introducing a stakeholder management methodology into the EU. *PMI® Global Congress 2009—EMEA.* Amsterdam.

Brückner & Swynnerton. (2014). A NEW ARCHITECTURE FOR AUTOMOTIVE HARDWARE-IN-THE-LOOP TEST. *ATZelektronik worldwide*.

Burzlaff et al. (2022). Semantic Interoperability Methods for Smart Service Systems: A Survey. *IEEE Transactions on Engineering Management*.

Castro et al. (2019). A Contemporary Vision of Project Success Criteria. *Brazilian Journal of Operations & Production Management*, pp. 66-77.

Celesti et al. (2017). Enabling secure XMPP communications in federated IoT clouds through XEP 0027 and SAML/SASL SSO. *Sensors (Switzerland)*.

Cestari et al. (2020). iPaaS in Agriculture 4.0: An Industrial Case. *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 48-53). Bayonne, France: IEEE.

Chatt, A. (2019, March 20). *What is event streaming?* Retrieved from IBM: https://www.ibm.com/blog/what-is-event-streaming-the-next-step-for-business/

Čolaković, A., & Hadzialic, M. (2018, July). Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, pp. 17-39.

Davis, K. D. (2023, September 5). *Stakeholder Analysis: Overview, Tools and Techniques*. Retrieved from knowledgehut.com: https://www.knowledgehut.com/blog/project-management/stakeholder-analysis

Desbiens. (2023). *Building Enterprise IoT Solutions with Eclipse IoT Technologies.* Embrun, Canada: Apress.

Dhall & Solanki. (2017). An IoT Based Predictive Connected Car Maintenance Approach. *International Journal of Interactive Multimedia and Artificial Intelligence (IJIMAI).*

Di Paolo et al. (2023). Security assessment of common open source MQTT brokers and clients.

Doran, G. T. (1981). There's a S.M.A.R.T. way to write managements' goals and objectives.

eJabberd. (n.d.). *ejabberd XMPP Server with MQTT Broker & SIP Service*. Retrieved from eJabberd.im: https://www.ejabberd.im/

Farhan et al. (2022). Efficient Data Transmission and Remote Monitoring System for IoT Applications.

Feiler et al. (2006). The Architecture Analysis & Design Language (AADL): An Introduction. *Software Engineering Institute.*

Fremantle. (2016). A Reference Architecture For The Internet of Things.

Ganzha et al. (2018). Towards semantic interoperability between internet of things platforms. *Internet of Things*.

Gartner. (2023, January). *Magic Quadrant iPaaS.* Retrieved from Gartner: https://www.gartner.com/doc/reprints?id=1-2CDRRLX3&ct=230126&st=sb

Gartner. (n.d.). *Definition of Business Technologist* . Retrieved from Information Technology Glossary: https://www.gartner.com/en/information-technology/glossary/business-technologist#:~:text=A%20business%20technologist%20is%20an,internal%20or%20external%20business%20use.

Gartner. (sd). *Gartner Magic Quadrant.* Retrieved from Gartner.com: https://www.gartner.com/en/research/methodologies/magic-quadrants-research

Gorman. (2023). *TCP vs UDP: Differences between the protocols*. Retrieved from Avast Academy: https://www.avast.com/c-tcp-vs-udp-difference#:~:text=TCP%20vs%20UDP%3A%20Differences%20between%20the%20protocols,reliable%20but%20works%20more%20quickly.

Grau et al. (2002). Architectural Description Languages and their Role in Component Based Design.

Gudoniene et al. (2023). The Scenarios of Artificial Intelligence and Wireframes Implementing in Engineering Education. *Sustainability 2023*.

Guth et al. (2016). Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture.

Hatmi, M. (.-d.-t.-p.-e.-r.-a.-r.-s. (2023, June 25). *Main Responsibilities and Required Skills for a Business Process Expert*. Retrieved from Spotterful: https://spotterful.com/blog/job-description-template/business-process-expert-responsibilities-and-required-skills

Heiler, S. (1995, June 1). Semantic interoperability. *ACM Computing Surveys*, pp. 271-273.

Hettig, N. (2021, January 14). *Connect Clients via MQTT to Apache Kafka*. Retrieved from Programmable Logic Controller (PLC) Next: https://www.plcnext-community.net/makersblog/connect-plcnext-control-via-mqtt-to-apache-kafka/

Hiraman et al. (2018). A Study of Apache Kafka in Big Data Stream Processing. *International Conference on Information , Communication, Engineering and Technology (ICICET).*

HiveMQ. (2019). *MQTT Topics, Wildcards, & Best Practices*. Retrieved from MQTT Essentials: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

Hmissi & Ouni. (2022). A Survey on Application Layer Protocols for IoT Networks. *International Journal on Advances in Telecommunications .*

Hornsby et al. (2009). XMPP-based wireless sensor network and its integration into the extended home environment. *The 13th IEEE International Symposium on Consumer Electronics* .

Iglesias-Urkia et al. (2019). Analysis of CoAP implementations for industrial Internet of Things: a survey. *Journal of Ambient Intelligence and Humanized Computing*.

Isode. (n.d.). *M-Link XMPP Server*. Retrieved from https://www.isode.com/products/m-link.html

Ivanovich. (2019). Languages of Architectural Description in Systems and.

John & Liu. (2017). A Survey of Distributed Message Broker Queues.

Josey. (2018). An Introduction to the TOGAF ® Standard, Version 9.2.

Jovanovic et al. (2021). Quarry: A User-centered Big Data Integration Platform. *Information Systems Frontiers*.

JSON for Linking Data. (n.d.). *JSON-LD Playground*. Retrieved from JSON for Linking Data: https://json-ld.org/playground/

Koziolek et al. (2020). A comparison of mqtt brokers for distributed iot edge computing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).*

Krüger et al. (2018). Flexible and reconfigurable data sharing for smart grid functions. *Energy Informatics*.

Laghari et al. (2021). Review and State of Art of Fog Computing. *Archives of Computational Methods in Engineering*.

Lakka et al. (2019). End-to-End Semantic Interoperability Mechanisms for IoT. *IEEE International Workshop on Computer-Aided Modeling, Analysis, and Design of Communication Links and Networks, CAMAD*.

Lankhorst et al. (2009). The Architecture of the ArchiMate Language. *Enterprise, Business-Process.*

Lauener & Sliwinsky. (2017). How To Design & Implement A Modern Communication Middleware Based On ZeroMQ. *ICALEPCS 2017 proceedings of the 16th International Conference on Accelerator and Large Experimental Physics Control Systems, October 8- 13, 2017, Barcelona, Catalunya.*

Liang et al. (2023). A Performance Study on the Throughput and Latency of Zenoh, MQTT, Kafka, and DDS.

Luckham, D. (1997, August 29). Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events. *Partial Order Methods in Verification*.

Maggi et al. (2022). A Security Analysis of the Data Distribution Service (DDS) Protocol. *Trend Micro Research.*

Maslow, A. H. (1970). Motivation and Personality (2nd ed.). *Harper & Row.* New York. Retrieved from Harper & Row.

Meertens. (2012). Mapping the business model canvas to ArchiMate. *SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*.

Mendelow, A. (1991). Stakeholder mapping. *Proceedings of the 2nd International Conference on Information Systems.* Camebridge.

Michigan Technological University. (2023). *What is Software Engineering?* Retrieved from Michigan Tech: https://www.mtu.edu/cs/undergraduate/software/what/#:~:text=of%20software%20applications.-,Software%20engineers%20apply%20engineering%20principles%20and%20knowledge%20of%20programming%20languages,the%20many%20career%20paths%20available.

MuleSoft. (2022). The Next Step in the Evolution of SOA. *API-led Connectivity*.

Murray-Webster & Simon. (2006). Making sense of stakeholder mapping.

Network-Centric Operations Industry Consortium - NCOIC. (2008, March 1). Systems, Capabilities, Operations,.

Nie et al. (2021). Design of big data integration platform based on hybrid hierarchy architecture. *Proceedings - 2021 IEEE 15th International Conference on Big Data Science and Engineering, BigDataSE 2021*.

Okoli, C. (2015, November 1). *A Guide to Conducting a Standalone Systematic Literature Review.* Retrieved from HAL Science: https://hal.science/hal-01574600/

Olan, M. (2003). UNIT TESTING: TEST EARLY, TEST OFTEN. *Computer Science and Information Systems*.

Paradkar. (2018). An iPaaS Reference Model. *Enterprise Architecture Professional Journal.*

Peffers et al. (2007, January). A design science research methodology for information systems research. *Journal of Management Information Systems*, pp. 45-77.

Priyadarshi & Behura. (2018). Analysis of Different IoT Protocols for Heterogeneous Devices and Cloud Platform. *2018 International Conference on Communication and Signal Processing (ICCSP).*

Rubin, R. S. (2002). Will the Real SMART Goals Please Stand Up?

Rudder & Main. (2020, August 25). *Scope Creep: Definition, Examples & How To Prevent It*. Retrieved from Forbes: https://www.forbes.com/advisor/business/scope-creep/

Ruparelia. (2010). Software Development Lifecycle. *ACM SIGSOFT Software Engineering Notes Volume 35.*

Sarrodie, J.-B. (2018). *Why Archimate*. Retrieved from Archi - ArchiMate Tooling: https://www.archimatetool.com/blog/2018/09/25/why-archimate/

Schneider & Abeck. (2023). Engineering Microservice-Based Applications Using an Integration Platform as a Service. *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE).*

Schneier & Bullock. (2020, August). *User Stories And The Alternatives*. Retrieved from Scrum INC.: https://www.scruminc.com/user-stories-and-the-alternatives/

Schwarz, L. (2022, November 3). *Efficiency vs. Effectiveness: What's the Difference?* Retrieved from Oracle / NetSuite: https://www.netsuite.com/portal/resource/articles/financial-management/business-efficiency-vs-effectiveness.shtml

Sein et al. (2011, March). Action Design Research. *MIS Quarterly*.

SFU Library. (n.d.). *Grey literature: What it is & how to find it*. Retrieved from https://www.lib.sfu.ca/help/research-assistance/format-type/grey-literature

Shalchian & Ravanmehr. (2016). INO340 telescope control system: middleware requirements, design, and evaluation. *Software and Cyberinfrastructure for Astronomy IV*.

Singh et al. (2021). Semantic Search System for Real Time Occupancy. *Proceedings of the 2021 IEEE International Conference on Internet of Things and Intelligence Systems, IoTaIS 2021.*

Singh, Van Sinderen & Wieringa. (2017). Reference Architecture for Integration Platforms. *Proceedings - 2017 IEEE 21st International Enterprise Distributed Object Computing Conference, EDOC 2017*.

Sobin. (2020). A Survey on Architecture, Protocols and Challenges in IoT. *Wireless Personal Communications*.

Sonnenberg & Vom Brocke. (2012). Reconsidering the Build-Evaluate Pattern in Design Science Research. *Design Science Research in Information Systems*, pp. 381-397.

Sridevi. (2014). User Interface Design. *International Journal of Computer Science and Information Technology Research*.

Stevanato et al. (2023). Virtualized DDS Communication for Multi-Domain Systems: Architecture and Performance Evaluation of Design Alternatives. *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*.

sysml.org. (n.d.). *What is the Systems Modeling Language (SysML)?* Retrieved from sysml.org.

Thomas & Fernández. (2008). Success in IT projects. *International Journal of Project Management*.

Tigase.net. (n.d.). *Tigase XMPP server*. Retrieved from https://tigase.net/xmpp-server/

Transforma Insights. (2023). *Current IoT Forecast Highlights.* Reading, United Kingdom.

Ulwick, A. W. (2005). *What Customers Want: Using Outcome-Driven Innovation to Create Breakthrough Products and Services.* McGraw Hill.

Uy & Nam. (2019). A comparison of AMQP and MQTT protocols for Internet of Things. *2019 6th NAFOSTED Conference on Information and Computer Science (NICS).*

Valderas & Torres. (2023). Towards a Semantic Interoperability in IoT Enhanced Business Processes. An Event-Driven Solution based on Microservices. *19th International Conference on Intelligent Environments, IE 2023 - Proceedings.*

Vermesan. (2018). Advancing IoT platforms interoperability.

Viswanath et al. (2016). System Design of the Internet of Things for Residential Smart Grid. *IEEE Wireless Communications*.

Vom Brocke & Siedel. (2012). Environmental Sustainability in Design Science Research: Direct and Indirect Effects of Design Artifacts. *Design Science Research in Information Systems*, pp. 294-308.

W3C. (n.d.). *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. Retrieved from World Wide Web Consortium: https://www.w3.org/TR/owl2-syntax/

Watkin & Koelle. (2016). *Practical XMPP.*

White, S. A. (2004). Introduction to BPMN.

Widell, N. (2020, July 23). *What is semantic interoperability in IoT and why is it important?* Retrieved from ericsson.com: https://www.ericsson.com/en/blog/2020/7/semantic-interoperability-in-iot

Wieringa & Morali. (2012). Technical Action Research as a Validation Method in Information Systems Design Science. *Design Science Research in Information Systems*, pp. 220-238.

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering.* Springer.

XMPP Organization. (n.d.). *XMPP Software*. Retrieved from XMPP: https://xmpp.org/software/

Xu et al. (2020). The Information Integration Platform for Satellite Earth Station based on the Dynamic Workflow Model. *ICEICT 2020 - IEEE 3rd International Conference on Electronic Information and Communication Technology.*

Yousuf & Asger. (2015, April). Comparison of Various Requirements Elicitation Techniques. *International Journal of Computer Applications*.

Yudidharma et al. (2023). A systematic literature review: Messaging protocols and electronic platforms used in the internet of things for the purpose of building smart homes. *Procedia Computer Science*.

Zehnder et al. (2020). StreamPipes Connect: Semantics-Based Edge Adapters for the IIoT. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Zhang et al. (2016). Prototype VOEvent network systems based on VTP and XMPP for the SVOM Chinese Science Center. *14th International Conference on Space Operations, 2016.*

Zippia. (2023, Juni). *What is an Intergration Specialist*. Retrieved from Zippia, The Career Expert: https://www.zippia.com/integration-specialist-jobs/

Zowghi & Coulin. (2006). Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In *Software Requirements Engineering and Managing.*

Appendix A: Elicited Requirements

This appendix consists of the elicited requirements of the project. A detailed description of the used methods is given. These requirements have been prioritized as can be seen in chapter 5 of the report.

Methods used for the Requirements

Eliciting Requirements

For requirements elicitation, Zowghi & Coulin (2006) gave an overview of different methods. The method which is selected to investigate is the Interview method. This method is described as "the most traditional and commonly used method for requirements elicitation" (Zowghi & Coulin, 2006). Three different types of interviews are given in the overview of Zowghi & Coulin. The first described method is the Unstructured Interview. This method follows the interview in a nature where the interviewer has limited control over the directions of the conversation. This limited control is caused because the interviewer does not follow a preselected agenda or set of questions. The risk of this method may involve that some topics may be neglected, or that the focus is too much on the detail of some areas. This method can be best applied to cases with limited understanding of the domain.

The second type of interview as mentioned by Zowghi & Coulin is the Structured Interview. This method follows a predetermined set of questions to gather the relevant information. This methods success depends on the knowledge about the right questions to ask, when to ask these questions, and who should answer these questions. These interviews could be guided by different structured interview templates. Structured interviews limit the investigation of new ideas because of the question bias by the interviewer, but this method can generally be seen as rigorous and effective.

The third type of interview as mentioned by Zowghi & Coulin is the Semi-Structured Interview. This method is a combination of both previous mentioned methods. Further, Yousuf & Asger (2015) added a more detailed description of this method. The Semi-Structured interview provides consistency with the help of structured questions, but also enables innovative ideas, topics, and themes with the help of the unstructured questions. The problems which are not treated in the structured part of the interview, will have a chance of being treated in the unstructured part of the interview. The downside is that this method can be time consuming.

The selected method for the requirement elicitation is the Semi-Structured Interview approach. This approach allows for flexibility, while also answering the important predefined questions.

Stating Requirements

For stating the requirements, three methods have been considered from Schneier & Bullock (2020). It was chosen to use the User Stories method within this research project. This method for formulating the requirements originated in software programming. This approach states who wants what, and why. User Stories focuses on the empathy for the end-users. According to Schneier & Bullock (2020), this method helps to understand what pain the customers are going through.

The format of user stories follows the "As a <type of user>, I want to <action>, so that I <achieve goal>" structure. This captures who the requirement is for, what they exactly want, and the desire why they want this requirement. An example user story for this project could be: "As an integration specialist, I want to seamlessly integrate different IoT appliances, so that I can make the integration of these appliances much easier, efficient, and effective."

To select the right format for requirements, the strengths and weaknesses of these stories have been considered. Schneier & Bullock (2020) gave an overview of these strengths and weaknesses of these stories.

For User stories, the strength is the thinking from the end-user view, putting the customer central. The developers understand the pain the users experience while using their product or service. The weakness of User stories can be that end users not always have the biggest role. Think of an API, in this case, a user story would be out of context. Schneier & Bullock (2020) gave the example of designing a brake paddle of a car, every user story will just be "As the driver of the car…". This will just make the user story not the best solution in every case.

Based on these strengths and weaknesses, it is chosen to use the User Stories for formulating the requirements in this project. This because this project puts the focus on the experience of the User in an iPaaS environment. Further, this method is also used by the company where this project is conducted, generating more sympathy for the project within the company.

Elicited Requirements

Integration Specialist

The first set of requirements have been elicited from an internal Integration Specialist. During the Semi-Structured Interview with this stakeholder, five functional requirements have been identified. These requirements have been formatted in the User Story format as described in the method section of this chapter. These User Stories are given below.

1. As an Integration Specialist, I want to be able to create IoT integrations in a low-code manner which supports the most common protocols, so that I can efficiently connect IoT devices to various systems without extensive coding efforts.
2. As an Integration Specialist, I want to have clear visibility over the solution with a dashboard that provides graphs and logging about integrations, so that I can monitor the performance and health of integrations in real-time and troubleshoot issues effectively.
3. As an Integration Specialist, I want to easily map/match the IoT data to the target data, so that I can ensure seamless data transfer between IoT devices and the intended destinations without data transformation hassles.
4. As an Integration Specialist, I want insights into the IoT messaging protocol that sends these messages, the target protocol receiving these messages, and the flow between them, so that I can understand the communication pathways and optimize integrations for efficiency and security.
5. As an Integration Specialist, I want a visual aid for creating the integrations with different sources and targets, so that I can intuitively design integration workflows, visually connect IoT devices with systems, and streamline the integration configuration process.

As described in the stakeholder analysis, this stakeholder can be seen as a Key Player. This involves that this stakeholder needs to be managed closely to make the project a success.

Business / End User

The second set of requirements have been elicited from a Business / End User working at sister organization Cape Groep. The business role of this stakeholder is a Business Consultant which consulted a Dutch postal company on a project involving a glove-embedded barcode scanner. During the Semi-Structured Interview with this stakeholder, nine requirements have been identified, stated below.

Functional Requirements:

1. As a Business User, I want to monitor integration failures and gain insights into the quantity of messages processed successfully, so that I can ensure the smooth operation of our IoT data integrations.

2. As a Business User, I want an alerting mechanism in place, such as email notifications, alerts, the ability to receive SMS or phone calls when critical integration issues occur, ensuring immediate response and resolution, so that I can be notified when critical integration issues occur, ensuring immediate response and resolution.
3. As a Business User, I need an integration debugger that allows me to examine messages when developing with the integrations, so that it is clearly visible how each message is handled to identify and address issues effectively.
4. As a Business User, I require IoT specific protocols (like MQTT in the Pro Glove project), so that seamless communication, data exchange, and interaction with various systems and devices is ensured.
5. As a Business User, I want to be able to reuse connections and explore a marketplace of integration solutions, so that I can get inspiration for integrations, leveraging existing resources and best practices to optimize our IoT data integrations.

Non-Functional Requirements:

6. As a Business User, I want that new components can be integrated rapidly and flexibly, so that issues can be resolved as quick as possible.
7. As a Business User, I want the integrations to have an appropriate reliability and availability for specific use cases, so that the clients working with the product experience no errors, ensuring a high level of service.
8. As a Business User, I want the iPaaS architecture to be scalable, so that I can anticipate on, and handle increased data loads efficiently.
9. As a Business User, I want the iPaaS solution to be user-friendly, so that it is accessible and usable by team members across different technical backgrounds.

As described in the stakeholder analysis, this stakeholder can be plotted in the Minimum Effort class. This involves that this stakeholder needs to be monitored during the project. This stakeholder may not be neglected, but the power of this stakeholder is limited.

Product Manager

The third set of requirements have been elicited from a Product Manager. This Product Manager also partly fulfils the role of Product Owner. The interviewee focusses on roadmaps, business cases, feature request, and strategy. During the Semi-Structured Interview with this stakeholder, seven requirements have been identified, stated below.

Functional Requirements:

1. As a Product Manager, I want to ensure that users can map their IoT data to the common data model by incorporating ontologies, so that users are able to have more structured and meaningful data integrations.
2. As a Product Manager, I want to ensure that our solution can handle the linking of large data models using ontologies while maintaining manageability, so that users maintain clarity and control over complex integrations, especially with large data volumes.
3. As a Product Manager, I want the solution to support semantics for sensor data coming from IoT hubs and other applications, so that I can enhance the capabilities of the solution.
4. As a Product Manager, I want the solution to incorporate a No code/low code solution that is model-driven and visually intuitive, so that users can configure and manage integrations without the need for extensive technical expertise.

5. As a Product Manager, I want the solution to be compatible with the definition of both run-time and design-time environments of eMagiz, so that seamless functionality throughout the entire integration development and execution process is ensured.
6. As a Product Manager, I want the solution to be compatible with semantic technologies like JSON-LD, so that users can expose or consume semantic data and link it to their data model seamlessly.

Non-Functional Requirements:

7. As a Product Manager, I want the solution to remain user-friendly and comprehensible, even with its complexity, so that the solution is accessible to a broad range of users.

As described in the stakeholder analysis, this stakeholder can be seen as a Key Player. This involves that this stakeholder needs to be managed closely to make the project a success.

Appendix B: Solution Design

This appendix consists of the Solution Designs of this project. A detailed description of the design is given. Further details are given in chapter 5 of the report.

Introduction

In this research project, the needs of stakeholders are expressed through User Stories, shaping the development of four distinct designs: Business Processes, Architectures, Sequence Diagrams, and User Interfaces.

The coherence among these designs is found in their interdependence and smooth transition from one design to the next. Beginning with User Stories, these narratives capture the diverse perspectives of the stakeholders, serving as the foundational guide for subsequent designs.

The shift from User Stories to Business Processes offers a higher-level view of the system's functioning. Business Processes outline the workflow and interactions among system components, aligning them with the user requirements.

Next, the Architecture designs build upon the Business Processes, focusing on the structural layout and technology components required to support the proposed processes. This ensures that the system architecture meets scalability, performance, and security needs while accommodating constraints or preferences established in earlier phases.

Following the Architectures, Sequence Diagrams act as a bridge between the abstract architecture and the functional behavior of the system. These diagrams illustrate component interactions in a chronological order, offering a dynamic view of the system across different scenarios.

Finally, the User Interfaces converts abstract concepts into user-friendly interfaces, tailored to user preferences and functionalities, ensuring a smooth user experience.

The coherence between these designs lies not only in their sequential order but in their interconnected nature. Each design builds upon insights gained from the previous one, guaranteeing that the end product meets stakeholder requirements and project objectives.

These designs are part of the research project which follow the following research questions:
(SQ1) How can IoT data integrations be incorporated in iPaaS architectures?

This question has been addressed through research focusing on integrating IoT into iPaaS architectures, forming the basis of the given designs.

(SQ2) How can semantic interoperability be incorporated in iPaaS architectures?

Similar to the first question, this inquiry has also been addressed through research on IoT integration in iPaaS architectures, forming the basis of the designs.

(SQ3) What are the key characteristics of an iPaaS architecture that leverages semantic interoperability to seamlessly integrate IoT data across different systems?

This question will be partly answered through the process and outcomes of the solution design, detailed later in the final report.

(SQ4) What can be the impact of iPaaS architectures that leverage semantic interoperability to seamlessly integrate IoT data across systems based on specific cases?

This question will be addressed by developing a Proof of Concept based on the solution design, showcasing the solution's impact through a case study.

User Stories

1. Source Schema Extractor:

1.1. As an Integration Specialist, I want to effortlessly extract the source schema of an IoT gateway or application so that I can seamlessly integrate it with other systems connected to the iPaaS.

1.2. As an Integration Specialist, I want a visually intuitive representation of the extracted source schema of an IoT gateway or application for a clear understanding of its structure within the iPaaS environment.

1.3. As an Integration Specialist, I want the capability to modify the extracted source schema of an IoT gateway or application so that I can rectify any discrepancies or inaccuracies in the schema representation.

2. Schema-Ontology Matcher:

2.1. As an Integration Specialist, I want the source schema to be automatically aligned with the Global Ontology of the integration environment so that there's seamless matching of the source system's schema with the iPaaS semantics.

2.2. As an Integration Specialist, I want a visual representation of the mapping between the source schema and the Global Ontology so that I can understand and validate the connection between these systems.

2.3. As an Integration Specialist, I want the ability to edit the mapping of the source schema to the Global Ontology so that I can ensure accurate alignment as per my comprehension.

3. Ontology Searcher:

3.1. As an Integration Specialist, I want to efficiently search for domain ontologies to enrich the Global Ontology so that I can incorporate the most contextually relevant semantics into the iPaaS.

3.2. As an Integration Specialist, I want the selected domain ontology to seamlessly integrate with the Global Ontology, ensuring automatic alignment with the iPaaS semantics.

3.3. As an Integration Specialist, I want the functionality to modify the mapping between the discovered domain ontologies and the Global Ontology so that I can ensure precise integration and alignment between these semantic elements.

4. Global Ontology:

4.1. As an Integration Specialist, I want a comprehensive visual overview of the Global Ontology so that I can gain a clear understanding of the semantic structure within the iPaaS environment.

4.2. As an Integration Specialist, I want the ability to edit and enhance the Global Ontology so that I can refine the semantic framework within the iPaaS environment.

5. Semantic Adapter:

5.1. As an Integration Specialist, I want an automated generation of a semantic adapter based on the mapped source schema to the Global Ontology so that I can smoothly deploy IoT integrations on the iPaaS's Run-time.

6. Low-Code:

6.1. As an Integration Specialist, I want to create IoT integration flows in a Low-Code manner so that I can easily and comprehensively manage and understand IoT integrations within the iPaaS.

Planning

The Development phase (4 weeks, weekly sprints) is planned as can be seen in the table below.

Sprint	User Stories
1	1.1, 1.2, 1.3, 4.1
2	2.1, 2.2, 2.3
3	3.1, 3.2, 3.3, 4,2
4	5.1, 6.1

Business Processes

The business processes are designed based on the given User Stories. The business processes are modelled in the Business Process Modelling Notation (BPMN) and are presented as images in the attachments.

BP1. Design-Time Process

The first designed process is based on a current Integration Cycle which consists of six phases. These phases are 1) Capture the Integration Requirements, 2) Design the Integration, 3) Create the Integration, 4) Deploy the Integrations, 5) Manage Integration, and 6) Improve the Integration. These are existing phases of an iPaaS platform. To enable such an iPaaS solution to enable Semantic Interoperability, an extension of the Design Phase is needed. Below, a new event is introduced called the "Create Semantic Adapter" event.

The first step in this event is to extract the source schema from the IoT gateway. This schema is then compared with a global ontology, which is unique for each integration environment but based on the same core IoT ontology.

If there is a complete match between the source schema and the global ontology, the integration specialist assesses the match. If the match is not perfect, a relevant ontology is retrieved from the semantic web. This retrieved ontology is then merged with the global ontology, and the integration specialist assesses the match again.

If the match is still not satisfactory, the process loops back to the schema matching step. If the match is successful, the global ontology is updated if necessary. After this, a Semantic Adapter is generated.

The Semantic Adapter is a set of transformation rules used by the Kafka Event Processor. These rules translate the incoming payload to match the global ontology of the environment, preferably in JSON-LD format.

This process ensures that the data from different IoT gateways can be integrated seamlessly into the platform, enhancing its interoperability and efficiency. It is a smart design that leverages the power of semantic web technologies to improve IoT integrations on iPaaS platforms.

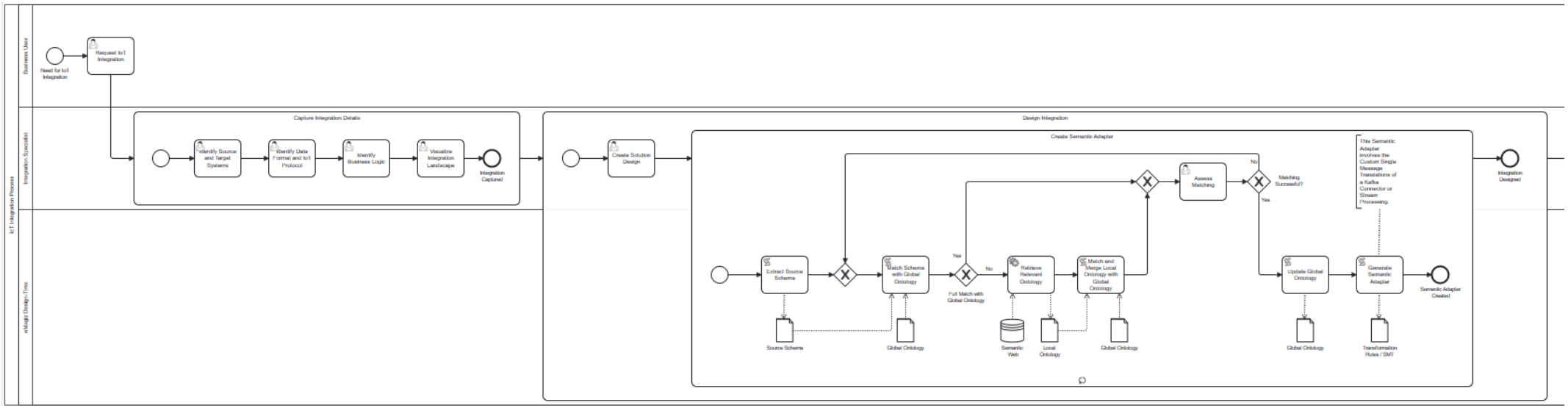

Figure 66: BP1. Design-Time Process (Part 1)

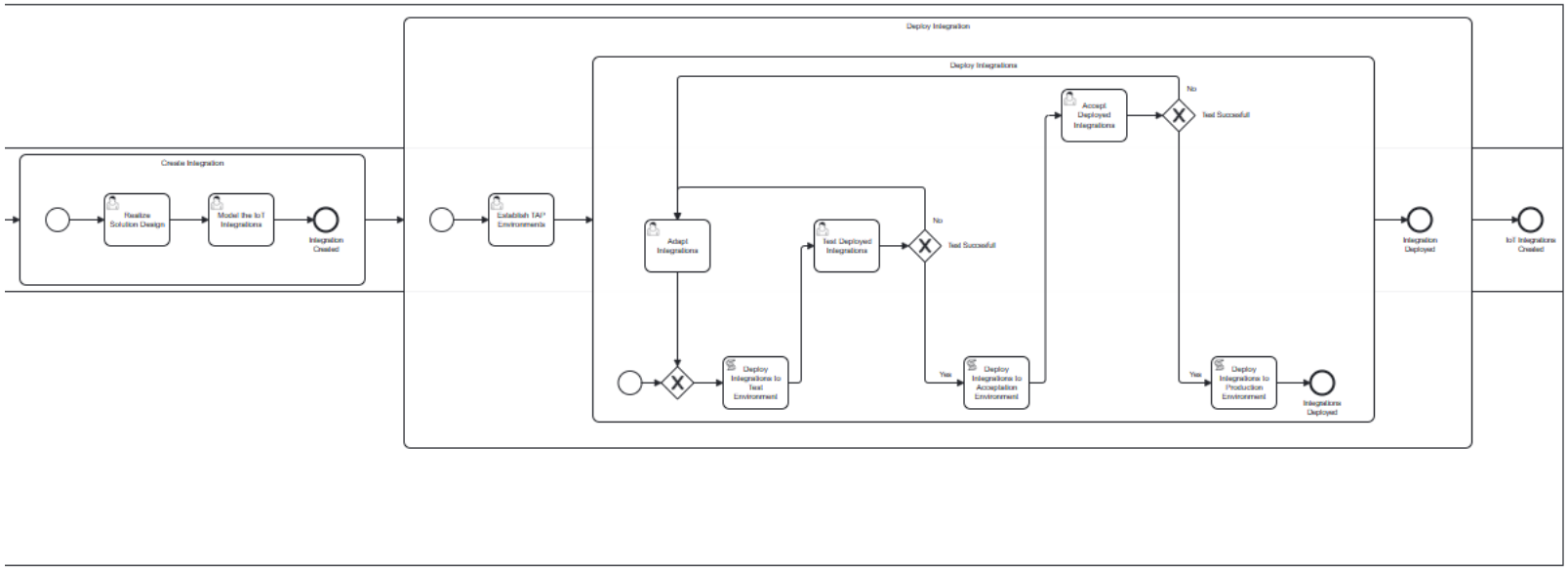

Figure 67: Design-Time Process (Part 2)

BP2. Run-Time Process

The second designed process is based on the Event Streaming pattern of Apache Kafka. This uses a Kafka Source Connecter, Kafka Stream Processor, and a Kafka Sink Connector to enable systems to publish and subscribe to the Kafka topics.

This process starts with an IoT Gateway that captures sensor data. This data is collected either when there is a change in the environment or at regular intervals. Once the data is captured, it is published to a topic on the MQTT proxy.

The MQTT proxy then forwards this payload to the Kafka Broker on the Kafka Source Connector. The Kafka Stream Processor first transforms the payload to the Global Ontology, following the Transformation Rules generated by the Semantic Adapter. After this transformation, the Stream Processor can execute other defined processing tasks on the payload.

Once the Stream Processor has finished processing, the payload is sent to the subscribers via the Sink Connectors of the Kafka Broker. All systems which are subscribed to the specific topic then receive the payload.

In terms of error handling, all components of the system, including the Kafka MQTT proxy, the Kafka Connector (Source & Sink), and the Kafka Event Stream Processor, log any errors they encounter. After these errors are logged, the Run-Time saves the messages in a database. The Business User is then notified of the error through a Dashboard Notification, SMS, or Mail.

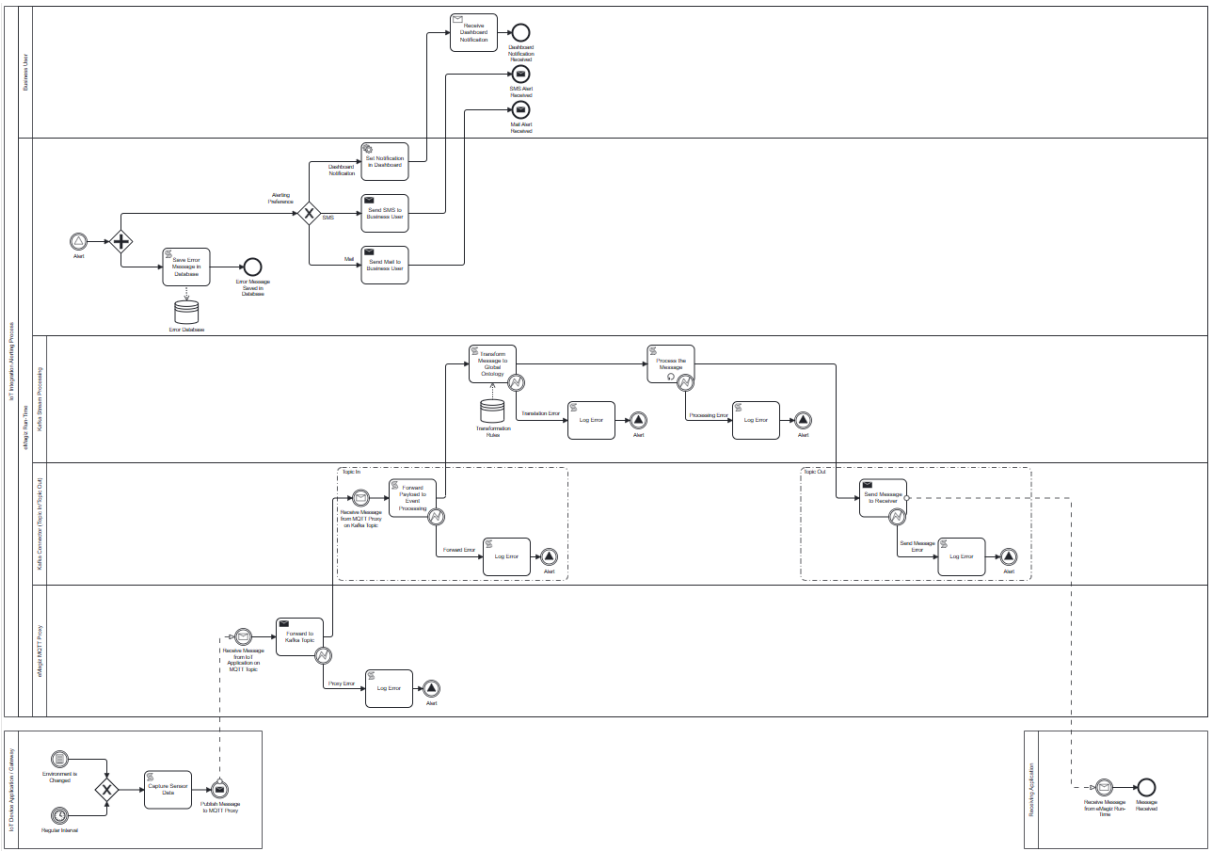

Figure 68: BP2. Run-Time Process

Architectures

The architectures are designed based on the given User Stories and processes. The architectures are modelled in the ArchiMate Architecture Description Language and are presented as images in the attachments.

A1. Design-Time

The proposed Design-Time architecture is an expansion of an existing iPaaS platform, designed to enhance the integration and interoperability of IoT gateways. The architecture is based on the Capture, Design, Create, Deploy, Manage, and Improve phases, with a particular focus on the Design phase.

In the Design phase, the Integration Modeller application component is enhanced with several new application functions. These include the Schema Extractor, Schema-Ontology Matcher, Ontology Searcher, Global Ontology Updater, and Semantic Adapter Generator.

The Schema Extractor is responsible for extracting the source schema of the IoT gateway. This extracted schema is then matched to the Global Ontology by the Schema-Ontology Matcher. The Ontology Searcher then searches the Semantic Web for relevant Best-Practice ontologies to expand the global ontology. Once these ontologies are identified, the Global Ontology Updater updates the global ontology. Finally, the Semantic Adapter Generator generates the Semantic Adapter, which consists of the Transformation Rules and Connectors that enable Kafka to semantically interoperate IoT sources.

This Design-Time architecture will be developed and run on Mendix in a Virtual Private Cloud. This ensures a secure and scalable environment for the architecture to operate in.

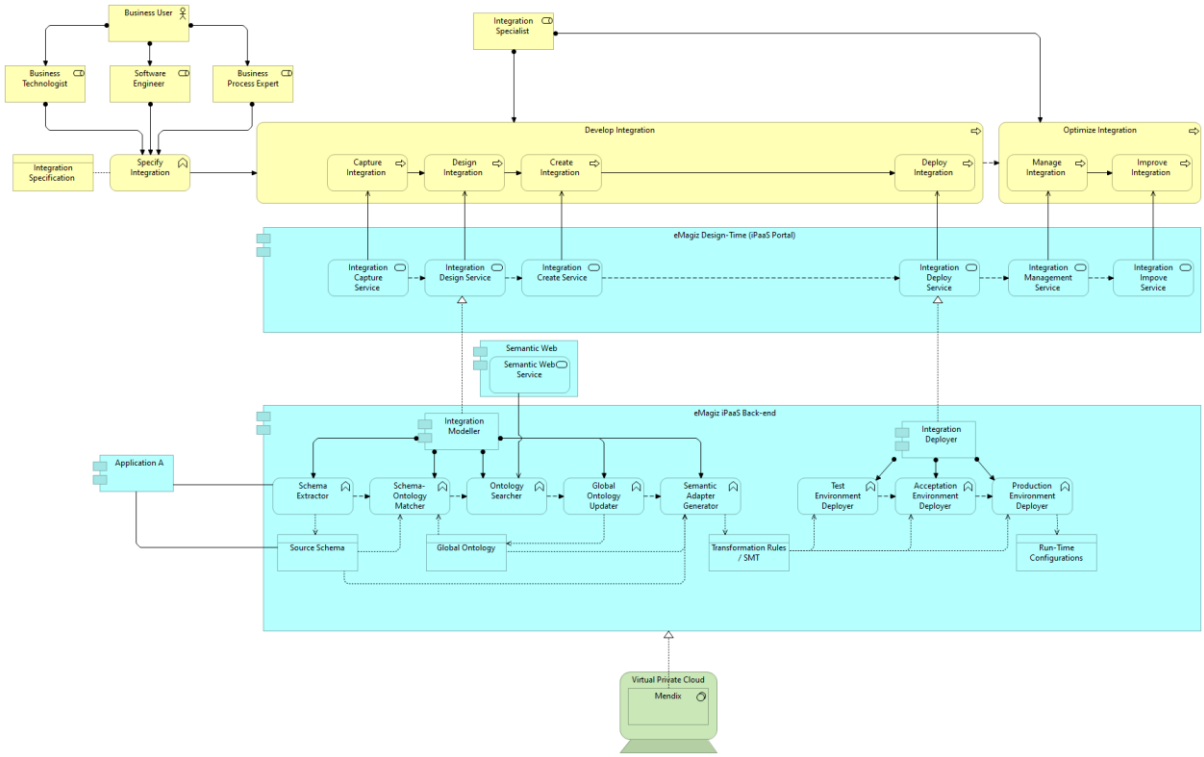

Figure 69: A1. Design-Time

A2. Run-Time

The proposed Run-Time architecture is an expansion of an existing iPaaS platform which uses Kafka as

Event Streaming Broker. The architecture is following the Event Streaming pattern of Kafka which involves a Source Connector, Event Stream Processor, and a Sink Connector.

The external IoT gateway application component publishes data to a topic on the Kafka MQTT Proxy application component using the MQTT protocol. The payload is next forwarded to the relevant Kafka Topic on the Kafka Source Connector (Topic In) interface of the Kafka Broker.

After the payload is received by the Kafka Broker, it is processed by the Kafka Event Stream Processor application service, which, based on the transformation rules, processes the payload. After the payload is successfully processed, it will be published to the subscribers by the Kafka Sink Connector (Topic Out) interface. These subscribers can be a variety of solutions, like another Application, or another Integration on the iPaaS environment.

This Run-Time architecture will be run on an Amazon Web Service (AWS) Managed Streaming Service for Apache Kafka (MSK) in a Virtual Private Cloud. This ensures a secure and scalable environment for the architecture to operate in. This AWS MSK will use the generated configurations to deploy the comprehensive solution. The Kafka Broker and Kafka Connect components will be running on this AWS MSK Virtual Private Cloud. The Kafka MQTT Proxy and the Kafka Stream Processor will be hosted on AWS Elastic Compute Cloud (EC2) instances.

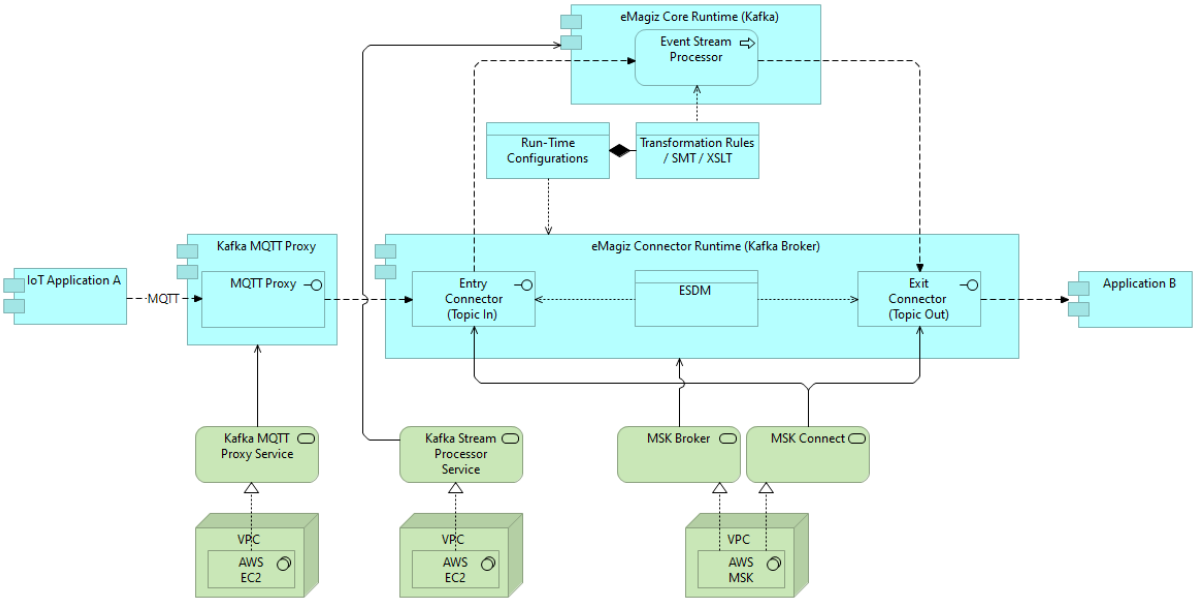

Figure 70: A2. Run-Time

Sequence Diagrams

The sequence diagrams are designed based on the given User Stories, processes, and architectures. The sequence diagrams are modelled in the UML-notation and are presented as images in the attachments.

SD1. Design-Time

The proposed Design-Time sequence diagram follows the interaction between different actors and system components.

The sequence starts with the Business User, presenting a Business Initiative for IoT Integration to the Integration Specialist. The specialist takes charge of the initiative and engages with the iPaaS for the development of IoT integration.

Upon activation, the iPaaS communicates with the IoT Source to Extract Schema, obtaining the Source Schema required for integration. Next, the iPaaS proceeds to Match the Source Schema with the Global Ontology, interacting with the Semantic Web to Crawl Relevant domain Ontologies. The identified Relevant Ontologies are then Merged and Matched with the Global Ontology within the iPaaS environment.

After this, the Environment Deployer is activated to Deploy Environments and Generate Run-Time Configurations. This deployment involves a Test Environment, ensuring the Integration Environment can be tested by the Integrations Specialist.

Continuing this sequence, another segment includes the deployment of an Acceptation Environment. Here, the Business User conducts an acceptation test within the Accepting Integration Environment, leading to the Accepted status communicated back to the Environment Deployer.

Subsequently, the final sequence involves the Deployment of Production Environment conducted by the Environment Deployer, resulting in the successful completion of the Design-Time sequence.

This sequence diagram defines the step-by-step progression of the Design-Time workflow, illustrating the orchestrated collaboration between stakeholders and technical components for the successful implementation of IoT integration using the iPaaS platform.

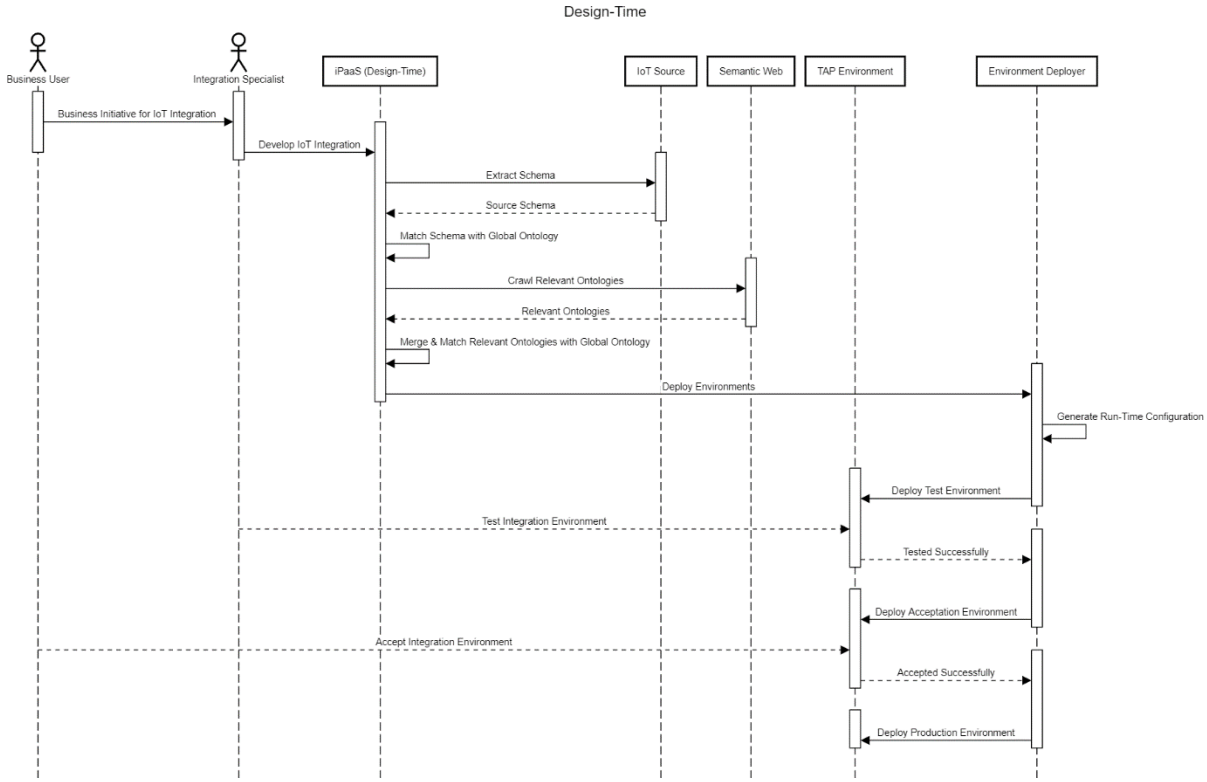

Figure 71: SD1. Design-Time

SD2. Run-Time

This Run-Time sequence diagram describes the interactions between the different system components during the Run-Time of the iPaaS, involving IoT devices and data processing through Kafka. Before the main branch of this sequence is executed, first the Consumers must subscribe to the Kafka Sink Connector (Topic Out) to receive the data in the end.

The sequence starts with the IoT Device, publishing sensor data with the help of one of the multiple communication protocols (Zigbee, Z-Wave, BLE, WiFi) to the IoT Gateway or IoT application.

Next, the IoT Gateway communicates with the Kafka MQTT Proxy to publish the sensor data to the MQTT Topic, facilitating the forwarding of data to the Kafka Source Connector (Topic In).

The Kafka Source Connector then processes the received payload and communicates with the Kafka Stream Processor to initiate payload processing. This involves transforming the data to comply with a Global Ontology, and executing other specified transformations to optimize the data before it is pushed to the subscribers.

Upon completion of processing, the Kafka Stream Processor sends the processed data back to the Kafka Sink Connector (Topic Out), which subsequently publishes this optimized data to the waiting Consumers.

This data flow and processing sequence within the Run-Time environment ensure the seamless processing and transformation of sensor data, conforming to a defined ontology, and delivering the refined data to the designated consumers.

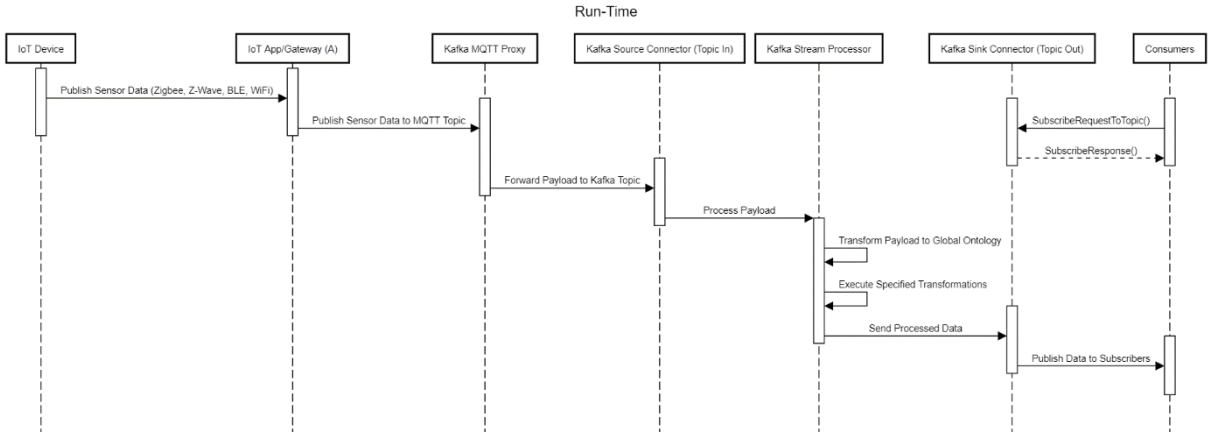

Figure 72: SD2. Run-Time

User Interfaces

The user interfaces are designed based on the given User Stories, processes, architectures, and sequence diagrams. The user interfaces are designed using wireframes and are presented as images in the attachments.

UI1. Capture – Requirements Designer

The first wireframe, the Requirements Designer, shows the available integration solutions in the left panel and visually represents the iPaaS in the larger right panel. Here, the Integration Specialist can drag the IoT Gateway onto the iPaaS canvas and create integration by drawing lines between the solutions and the iPaaS.

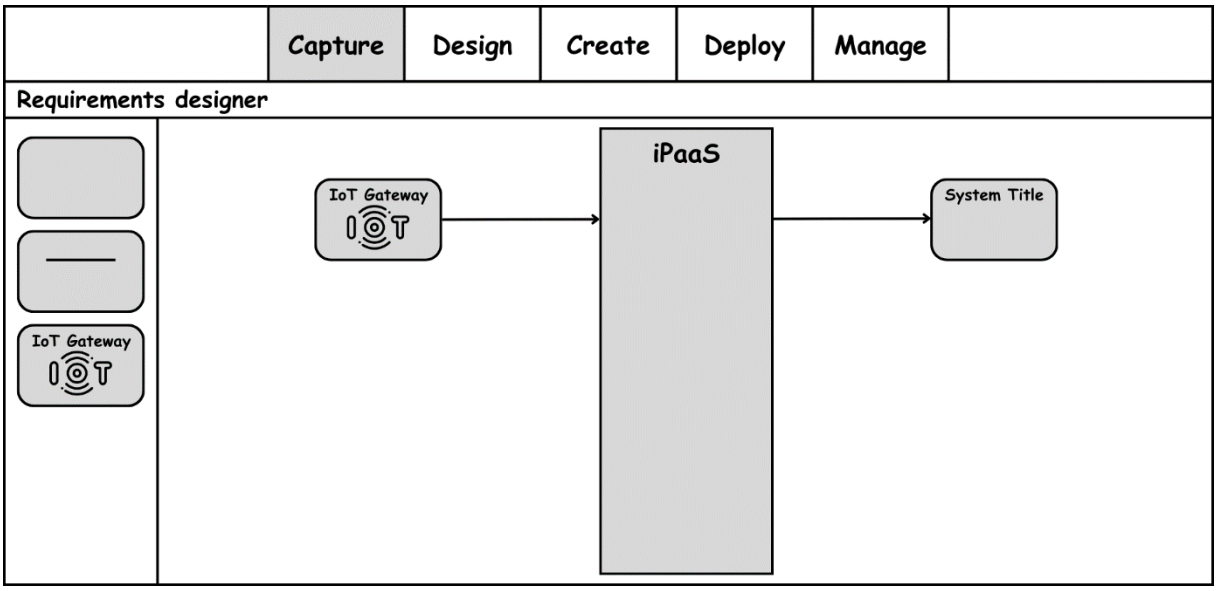

Figure 73: UI1. Capture – Requirements Designer

UI2. Capture – Edit IoT Gateway

Upon opening the IoT Gateway solution, the Integration Specialist can modify specific elements like Gateway names, message formats, the MQTT proxy host, topics & devices, and the task state of the integration.

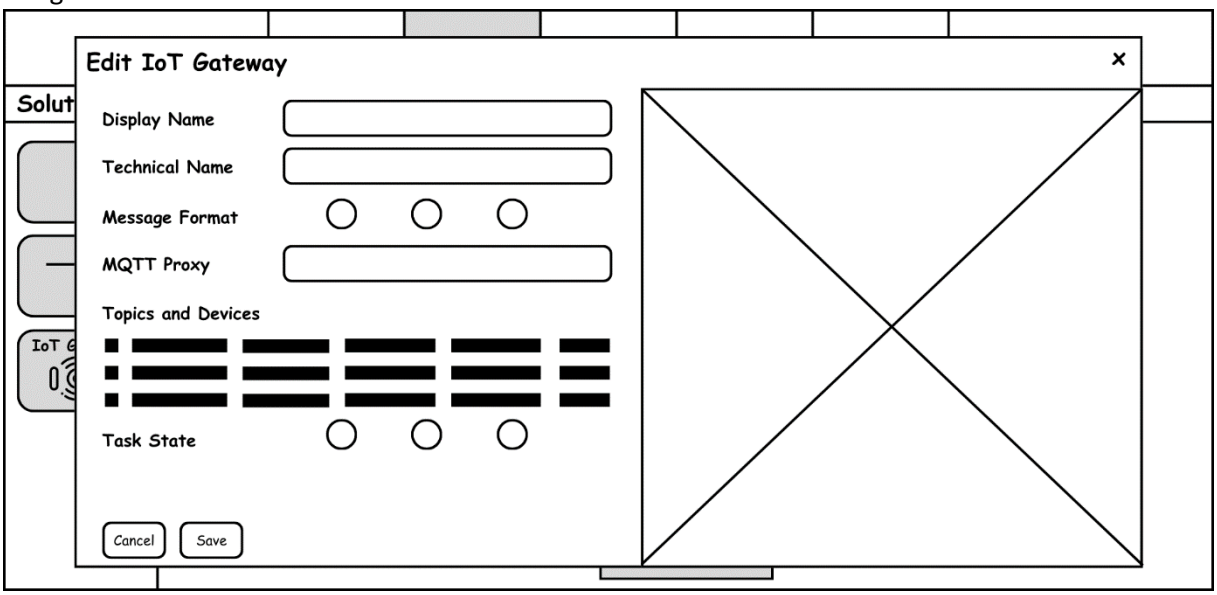

Figure 74: UI2. Capture – Edit IoT Gateway

UI3. Design – Solution Designer

Following the capture, the Integration Specialist moves to design the integration. This stage involves defining the source schema of the IoT gateway, particularly through the Extract Schema function, allowing customization and schema creation.

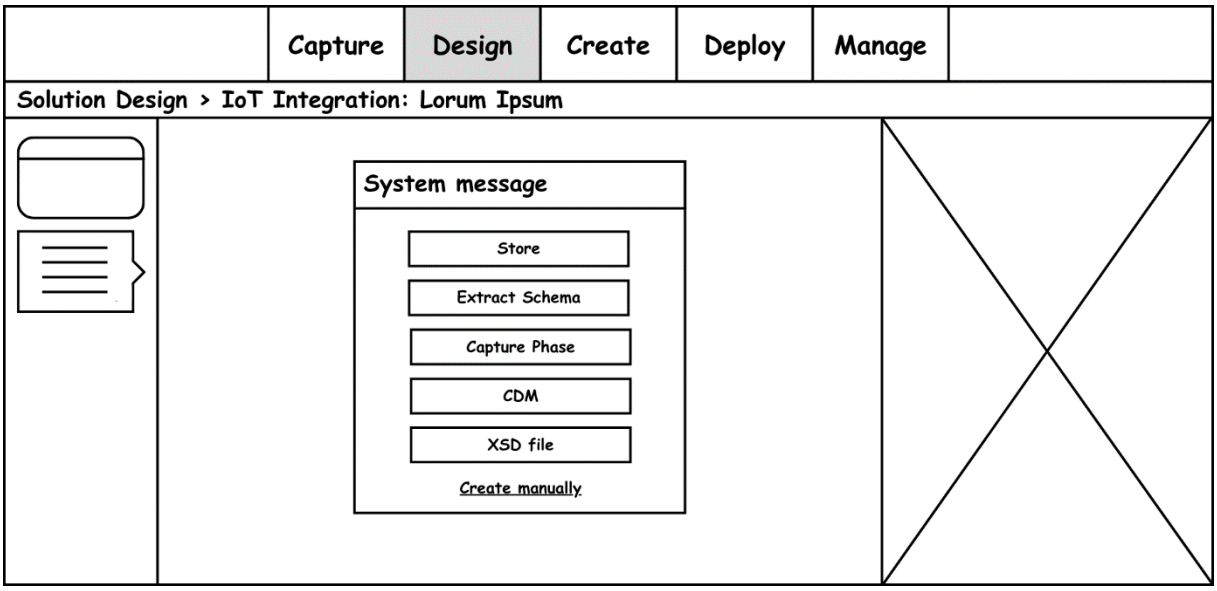

Figure 75: UI3. Design – Solution Designer

UI4. Design – Waiting for incoming payload

The Extract Schema function prompts the iPaaS to connect to the designated MQTT proxy, receiving payloads from the IoT gateway, thereby initiating the schema extraction process.

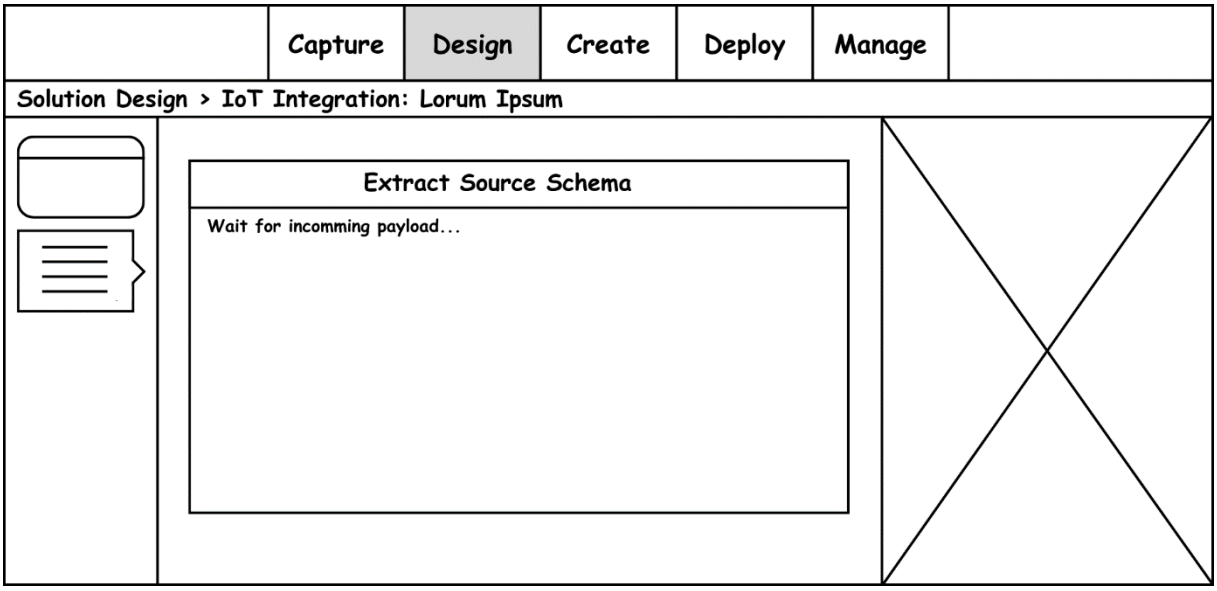

Figure 76: UI4. Design – Waiting for incoming payload

UI5. Design – Received Data

Post-receiving the payload, the Integration Specialist can view and edit the payload in JSON format based on their expertise. This modified schema needs to be saved for further extraction.

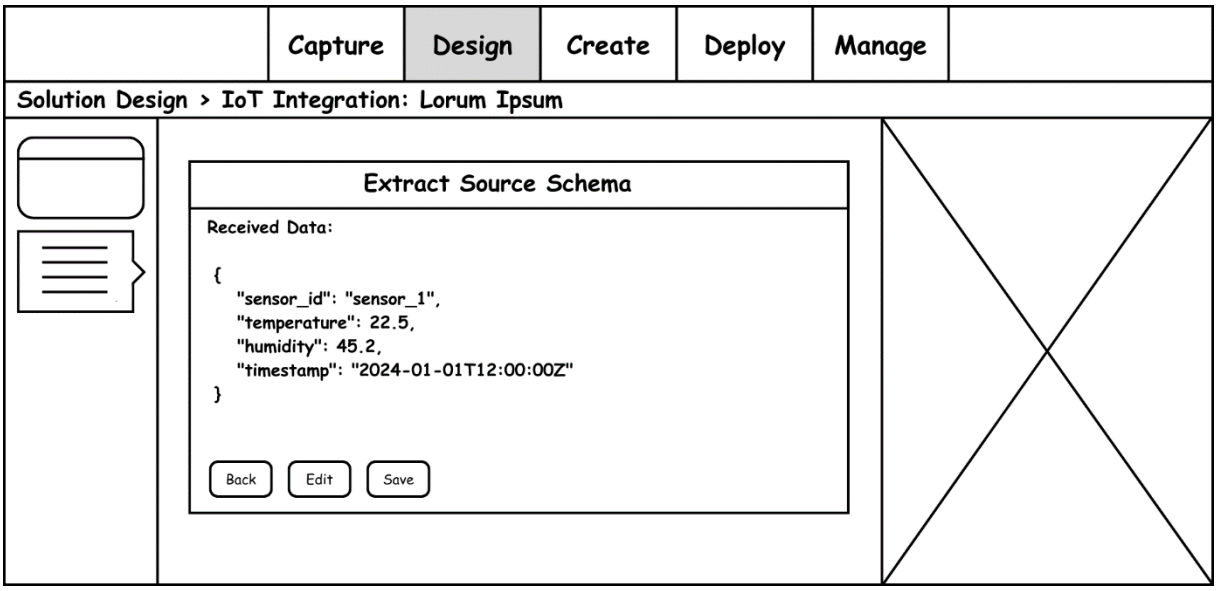

Figure 77: UI5. Design – Received Data

UI6. Design – Extracted Schema

Upon saving the payload, the schema is extracted, visually represented as a model with entities, attributes, and relationships. The Specialist can further edit this model to match their knowledge, after which it must be mapped to the Global Ontology.

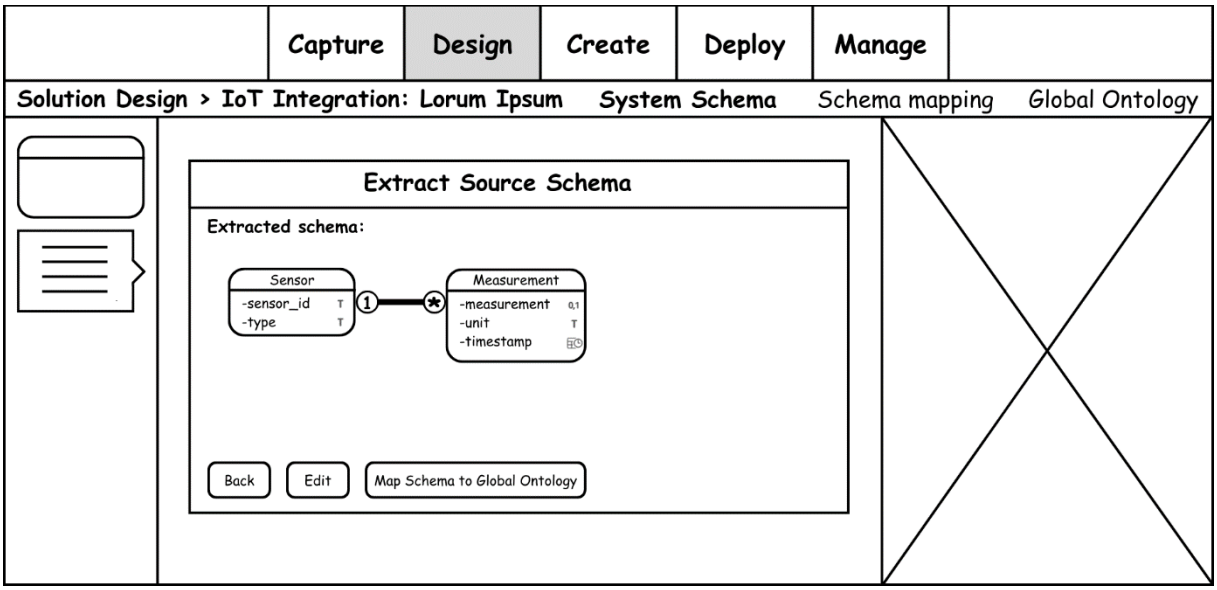

Figure 78: UI6. Design – Extracted Schema

UI7. Design – Schema Mapping

Mapping the schema to the Global Ontology creates an overview displaying entities and their attributes, in relation to their corresponding classes. This matching process is editable, and if the Global Ontology lacks coverage, a Domain Ontology can be added.

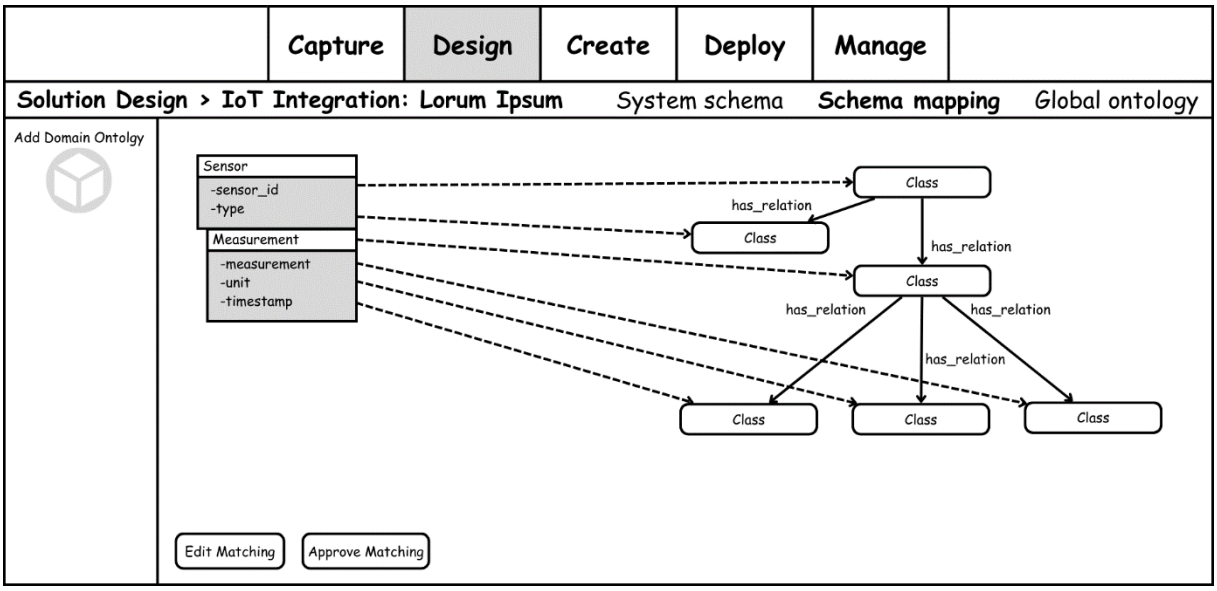

Figure 79: UI7. Design – Schema Mapping

UI8. Design – Add Domain Ontology

Adding a Domain Ontology involves selecting relevant ontologies, aligning them with the Global Ontology, and updating this Global Ontology. The source schema will be rematched to the updated Global Ontology, which needs approval or further editing by the Integration Specialist.

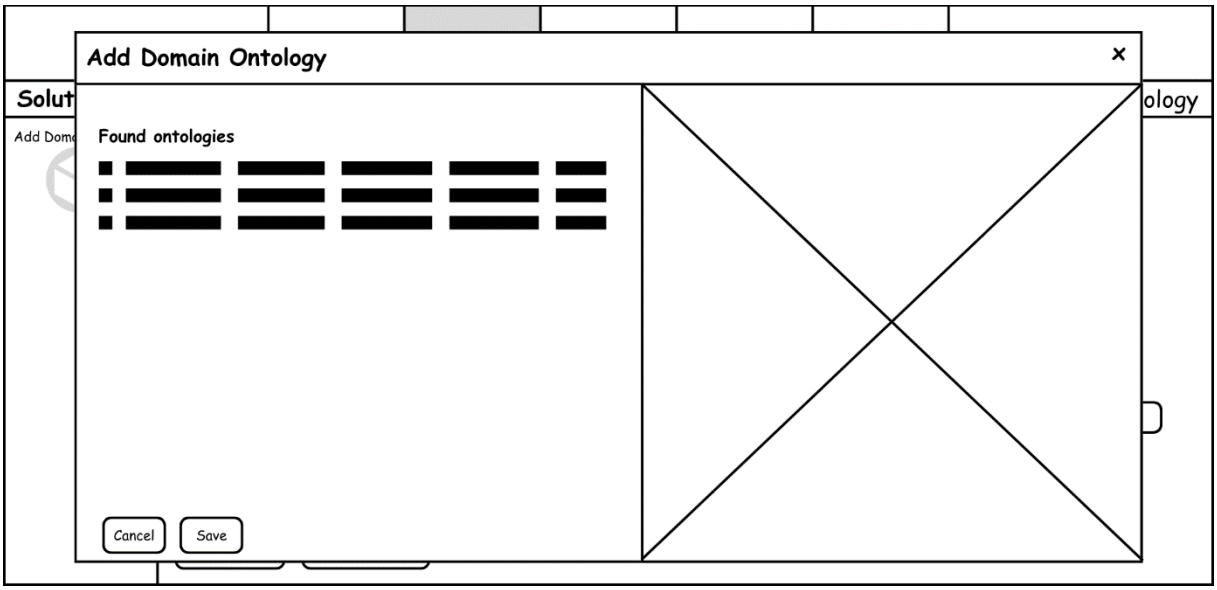

Figure 80: UI8. Design – Add Domain Ontology

UI9. Design – View Global Ontology

Upon approval, the Global Ontology is visible, presenting any extensions made by Domain Ontologies. This visual representation includes the Core IoT Ontology extended with relevant Domain Ontologies.

Once the schema-to-Ontology mapping is approved, a Semantic Adapter is generated in the background. This adapter includes transformation rules for IoT schemas into JSON-LD, configured with MQTT Proxy settings. Additionally, data transformation based on CDM/ESDM specifications from the Global Ontology is also facilitated.

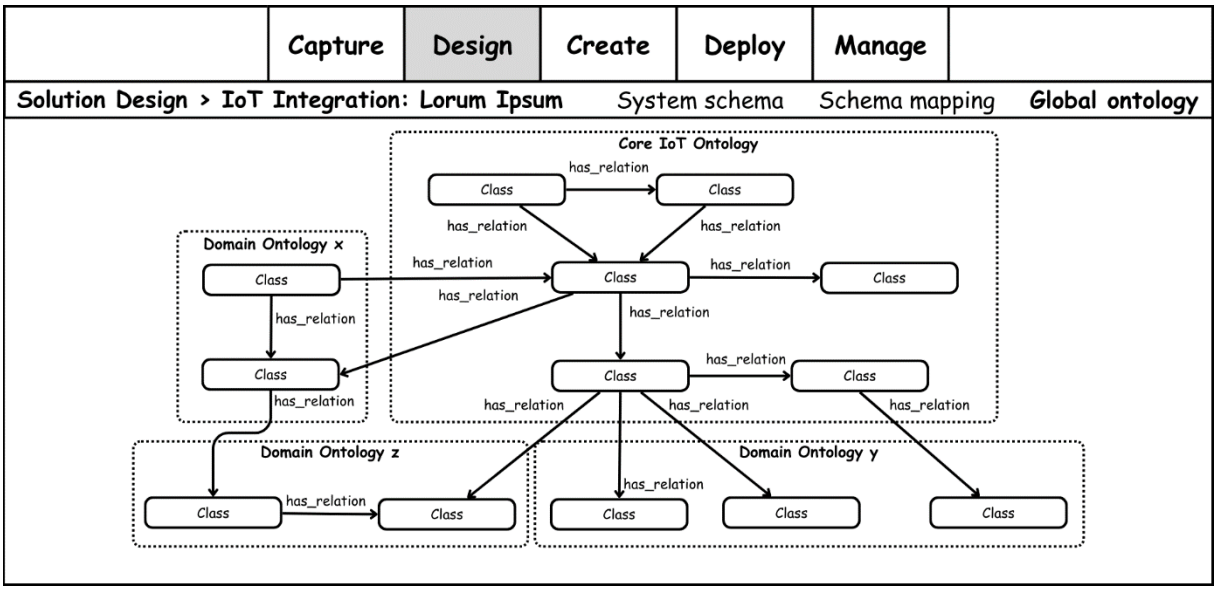

Figure 81: UI9. Design – View Global Ontology

UI10. Create – Processes

Finally, the created transformations can be used in the Integration Flow designer of an iPaaS portal. Two types of transformations are generated, the Source to JSON-LD, and the Source to CDM/ESDM. The first one can be used to expose the IoT data in a semantic manner to other systems in the iPaaS environment. The other transformation can be used to match the incoming data to the CDM/ESDM of the integration environment, after which it can be used in other integration flows or applications.

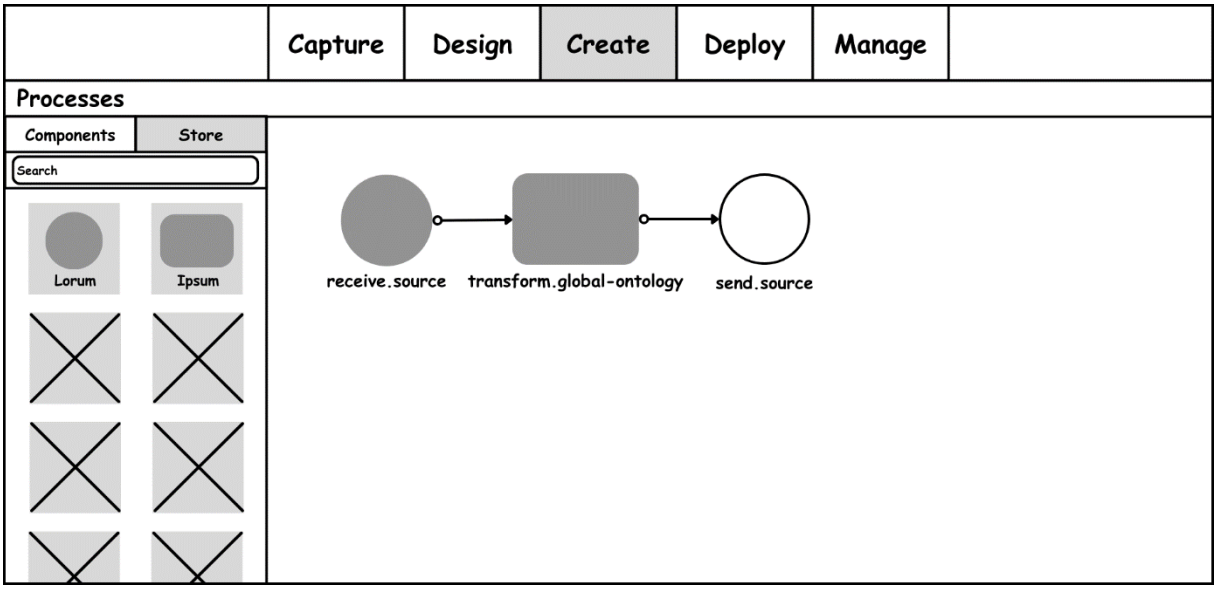

Figure 82: UI10. Create – Processes

Appendix C: Screenshots of the Proof of Concept

This appendix consists of the captured screenshots from the developed Proof of Concept. A short description of the screenshot is given in the caption of the figures. A more detailed description is given in chapter 6 of the report.

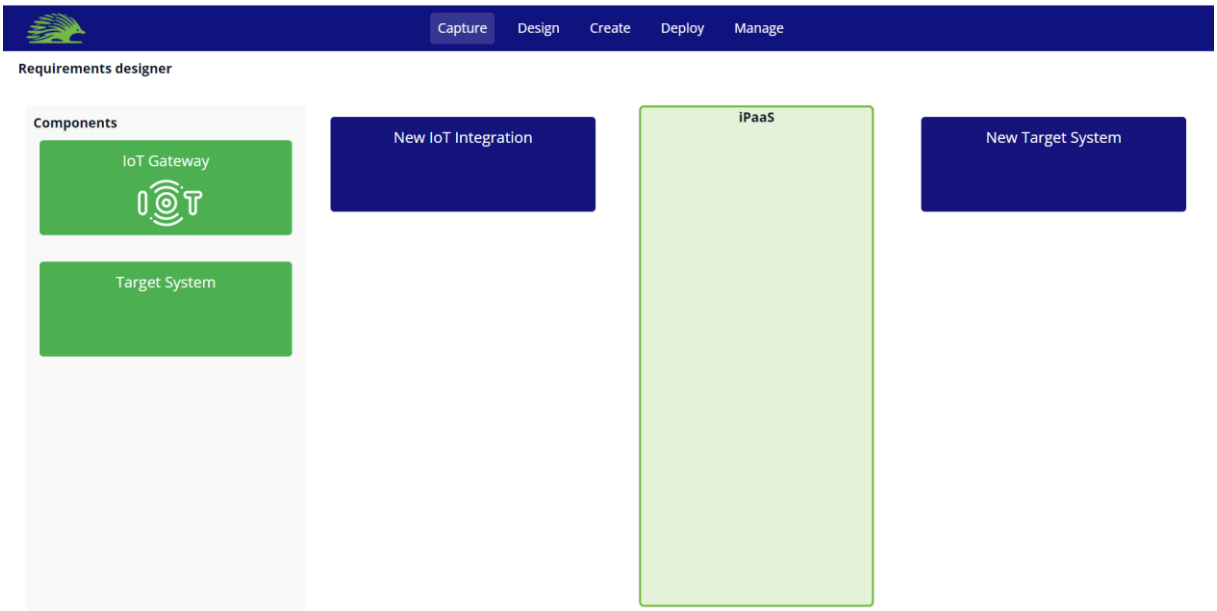

Figure 83: Overview of Capture Phase

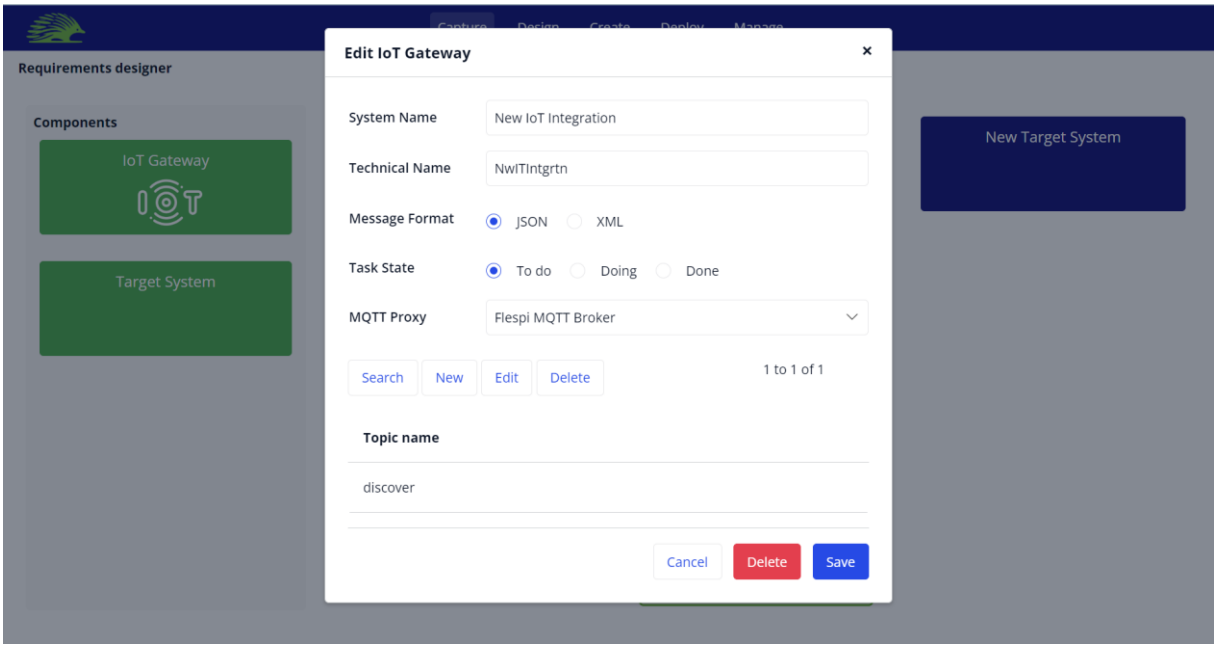

Figure 84: Creating an IoT Integration

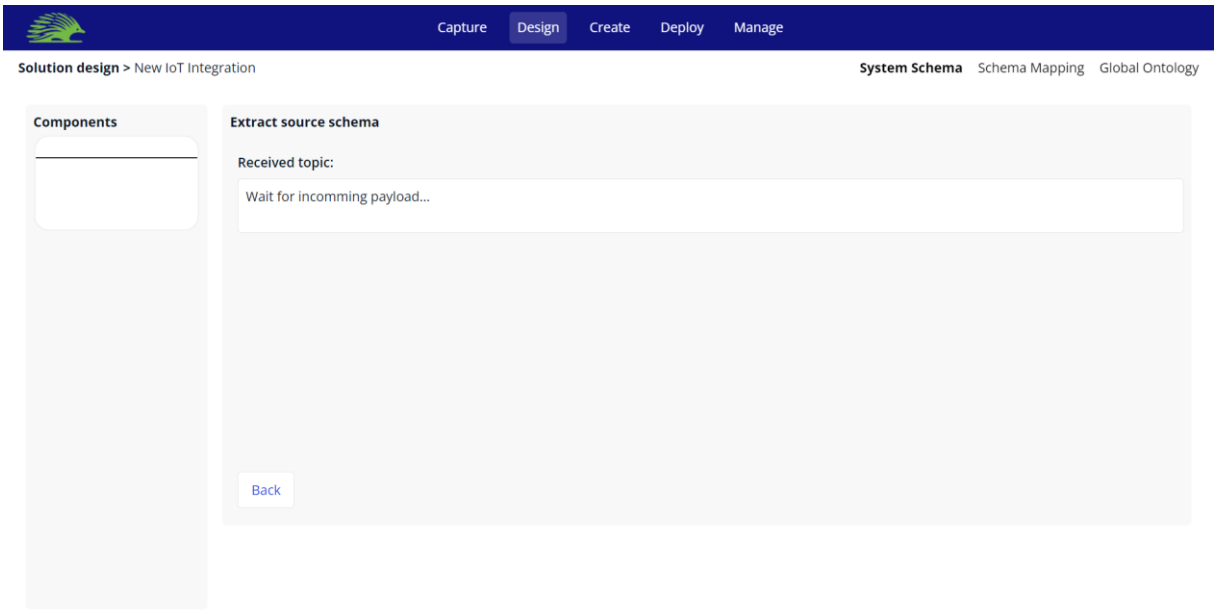

Figure 85: Waiting for Payload to be Published on the MQTT Broker

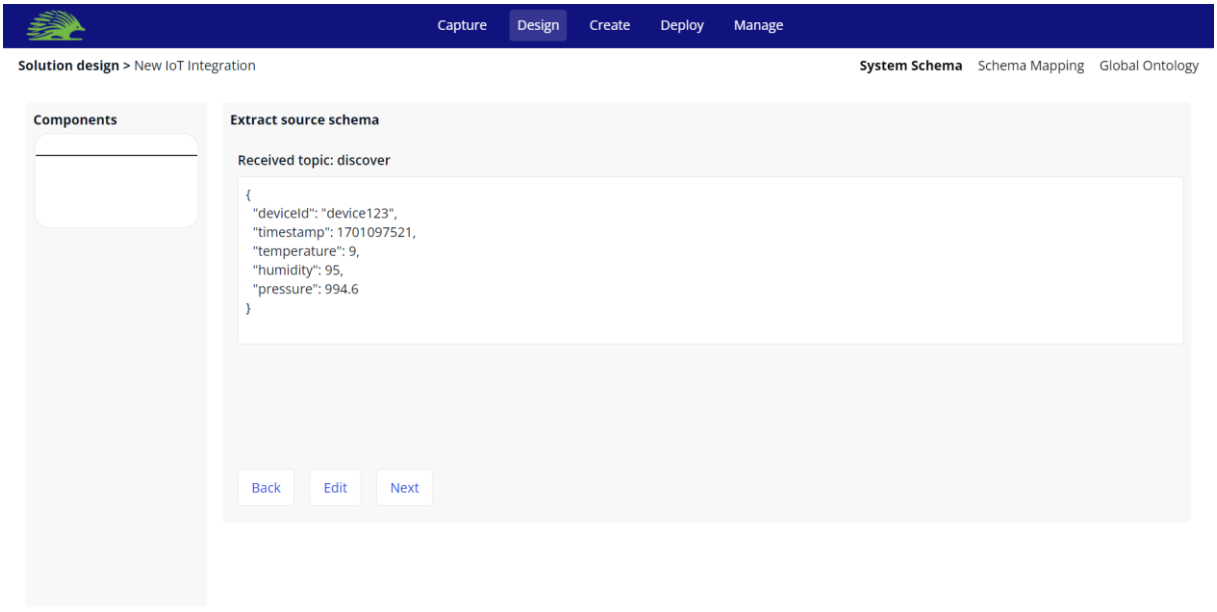

Figure 86: Captured Payload from the MQTT Broker

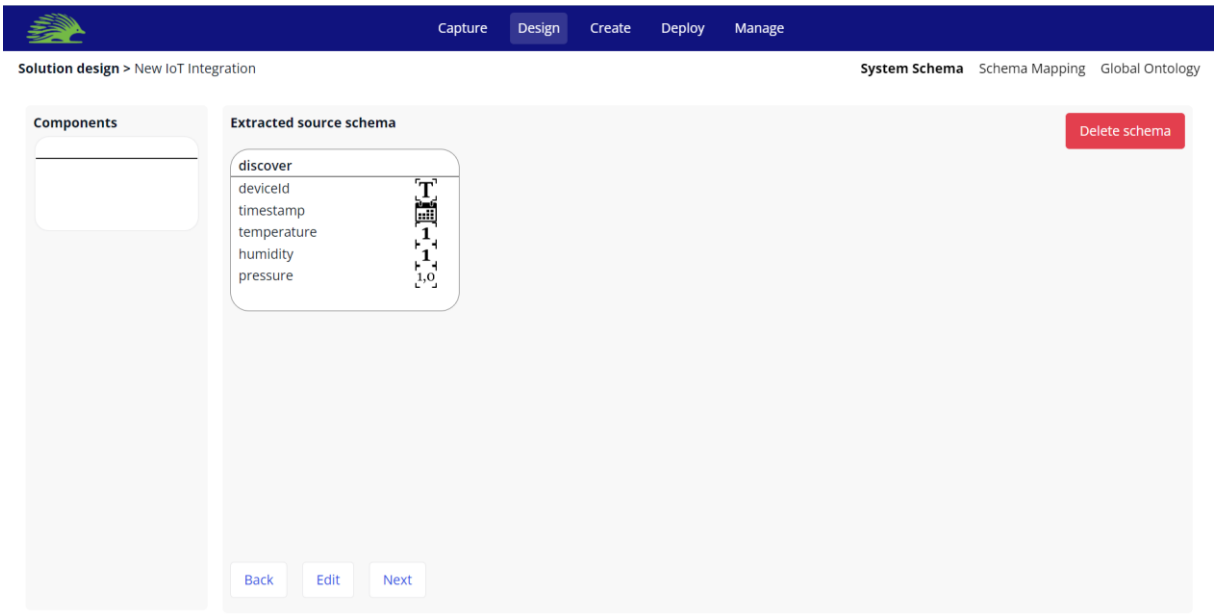

Figure 87: Extracted Message Schema from the Payload

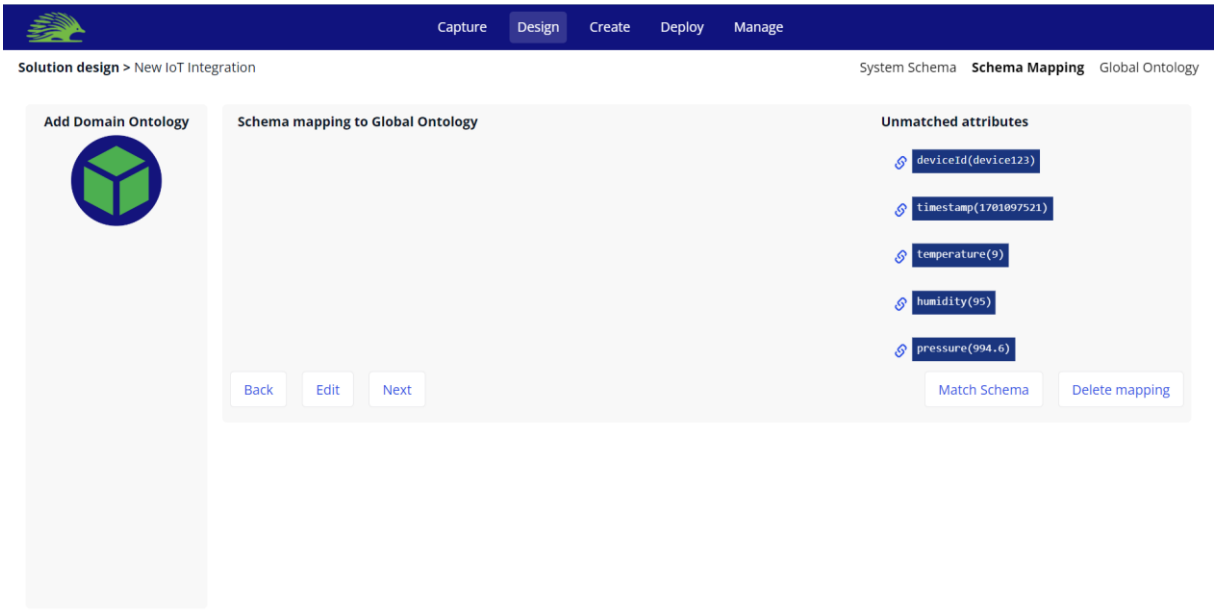

Figure 88: Start of Mapping the Source Schema to the Global Ontology

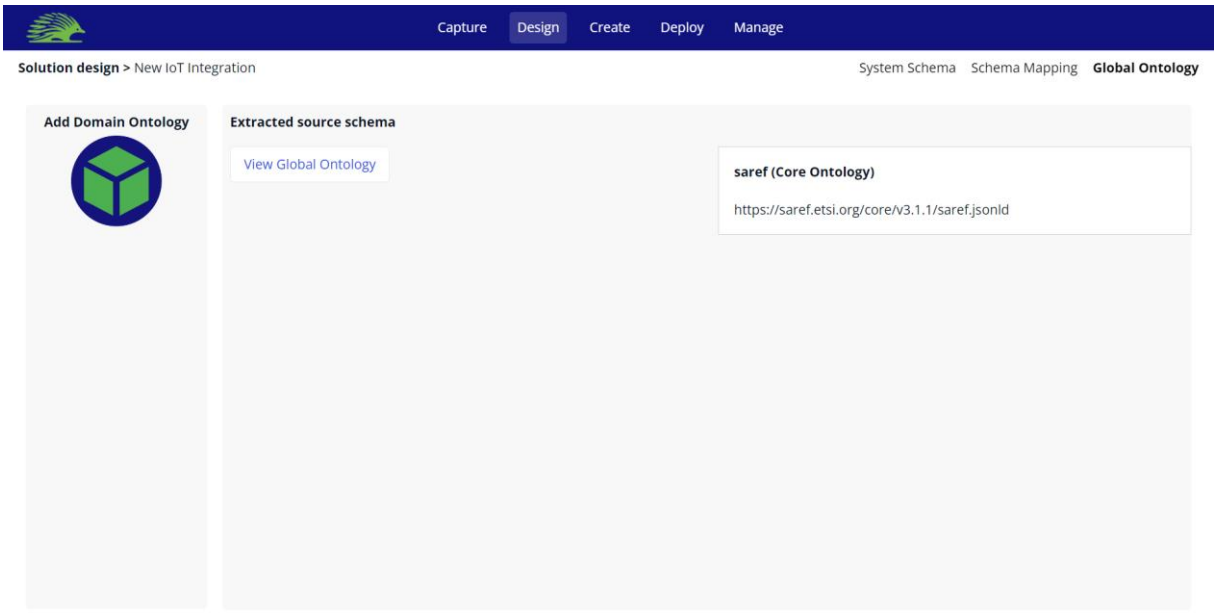

Figure 89: Defining the Global Ontology

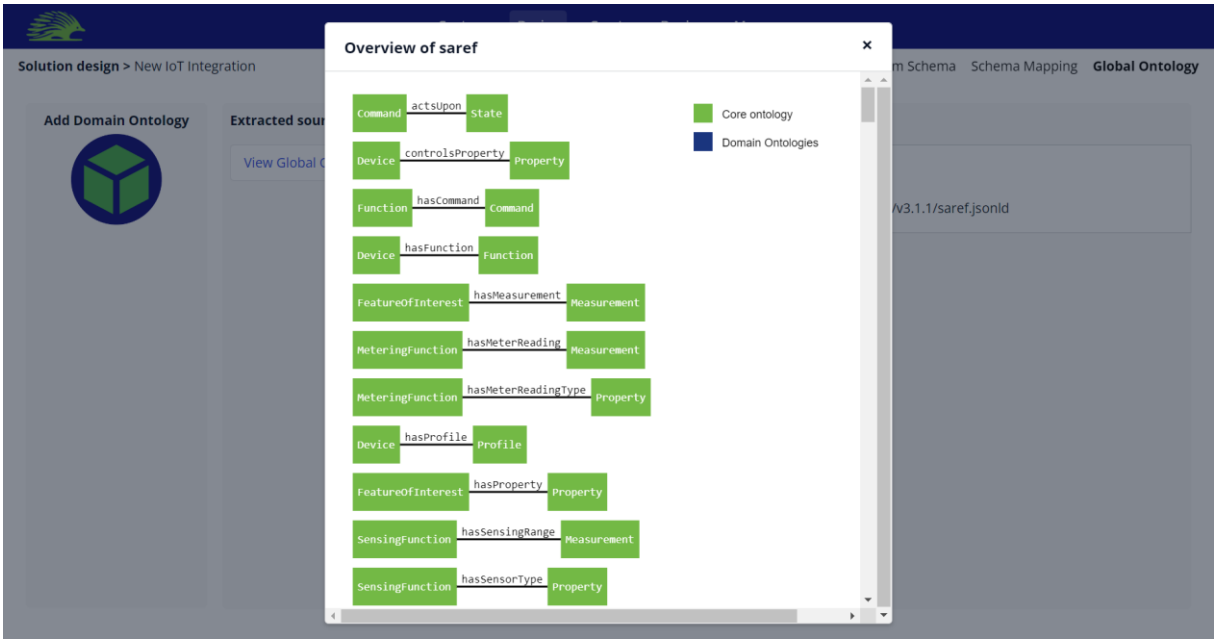

Figure 90: Overview of the Global Ontology

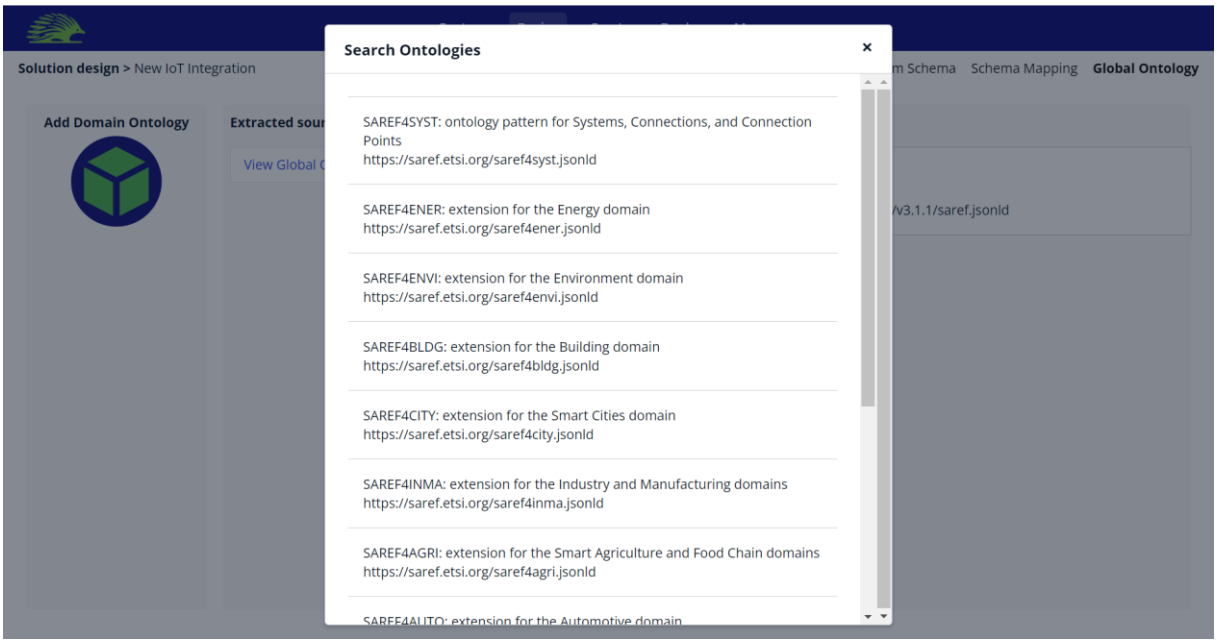

Figure 91: Search for Domain Ontologies

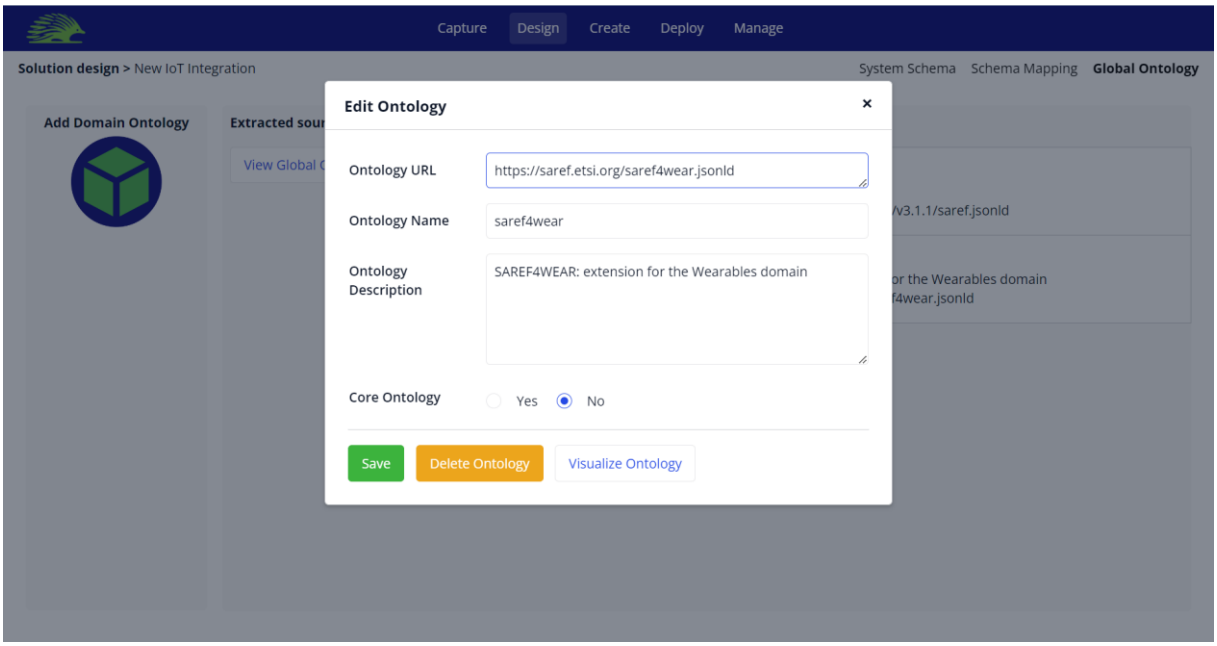

Figure 92: Import the Selected Domain Ontology

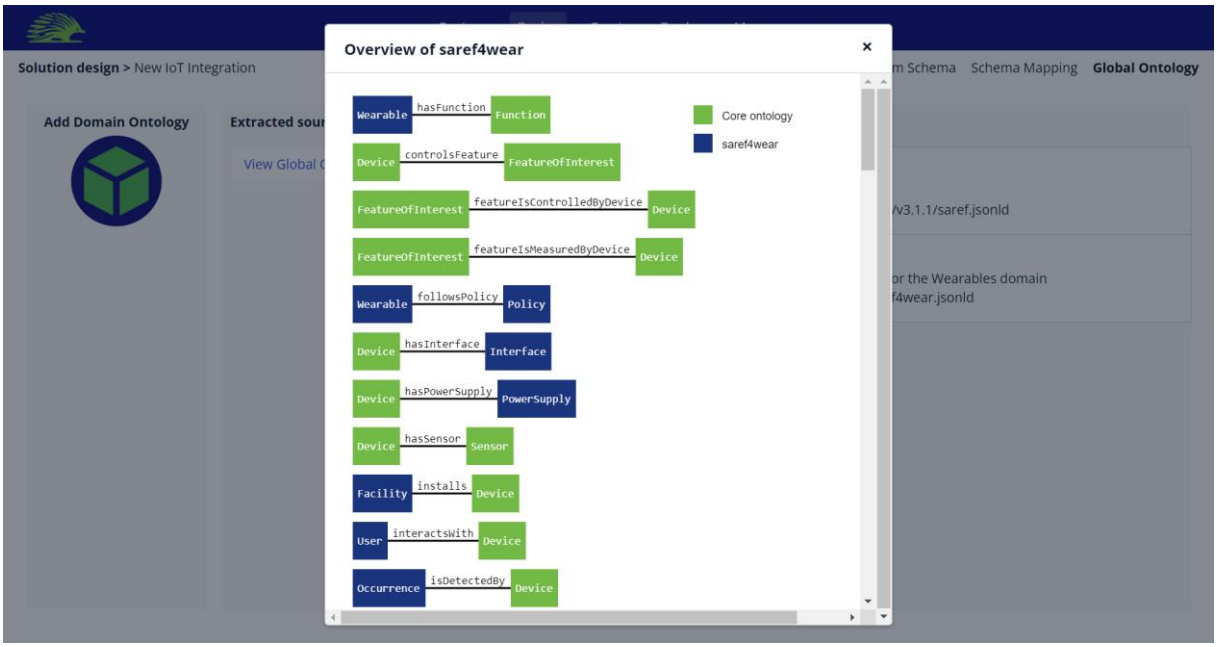

Figure 93: Visualize the Imported Domain Ontology

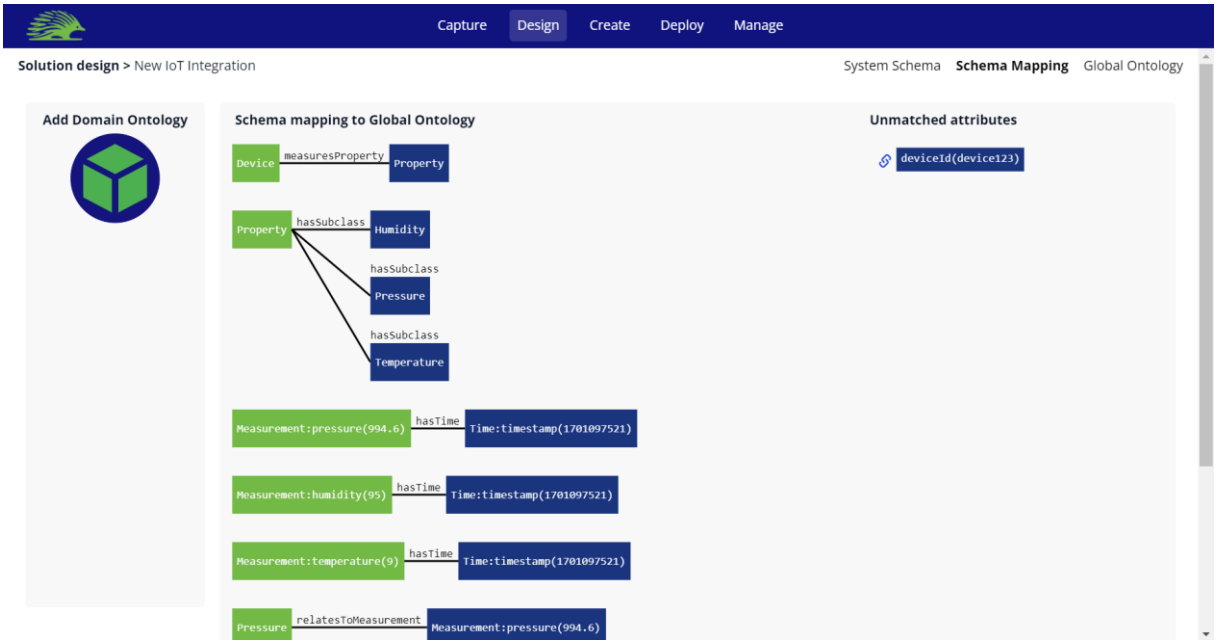

Figure 94: Mapping of the Source Schema to the Global Ontology

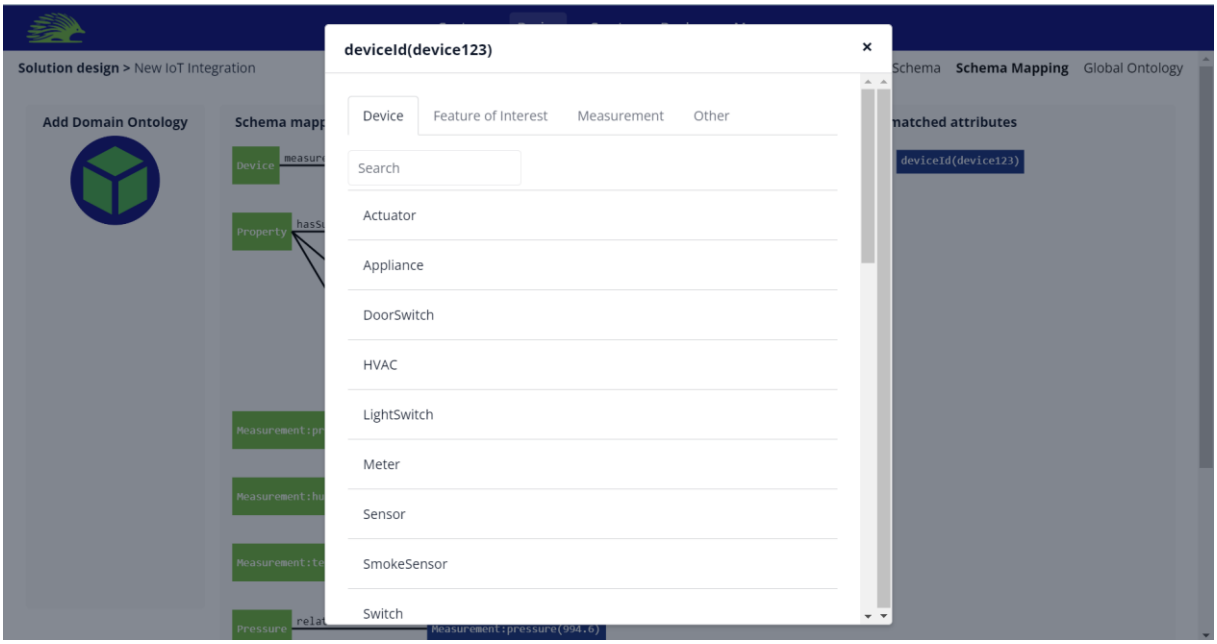

Figure 95: Manually Mapping a Source Schema Attribute to a Class of the Global Ontology

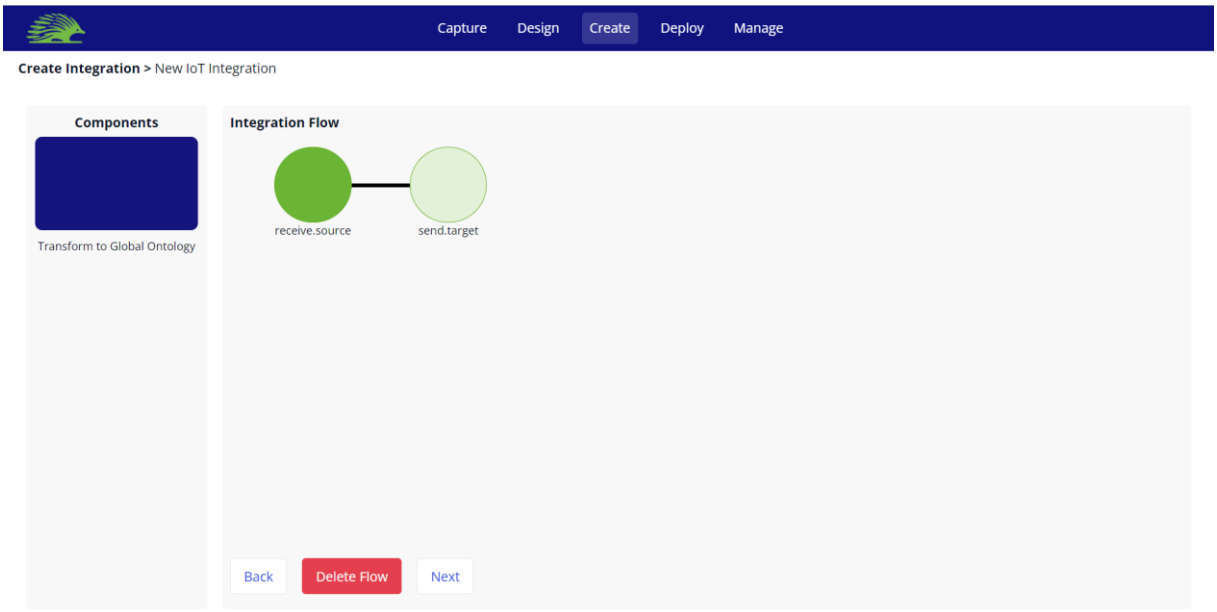

Figure 96: Creating an Integration Flow

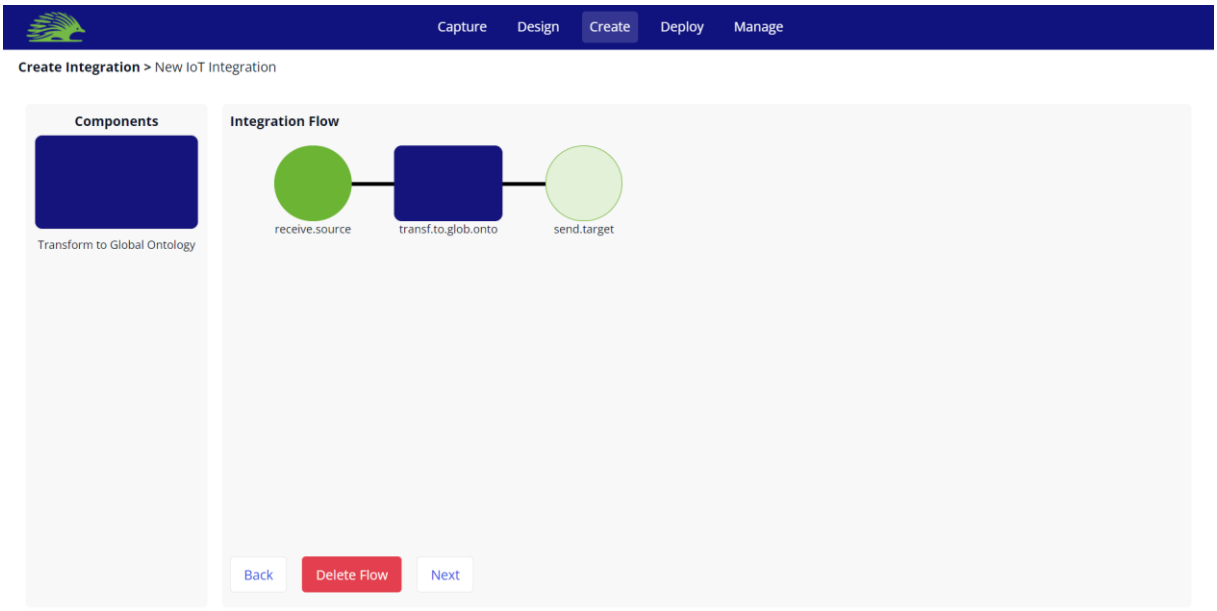

Figure 97: Added Integration Flow Transform to Global Ontology Event

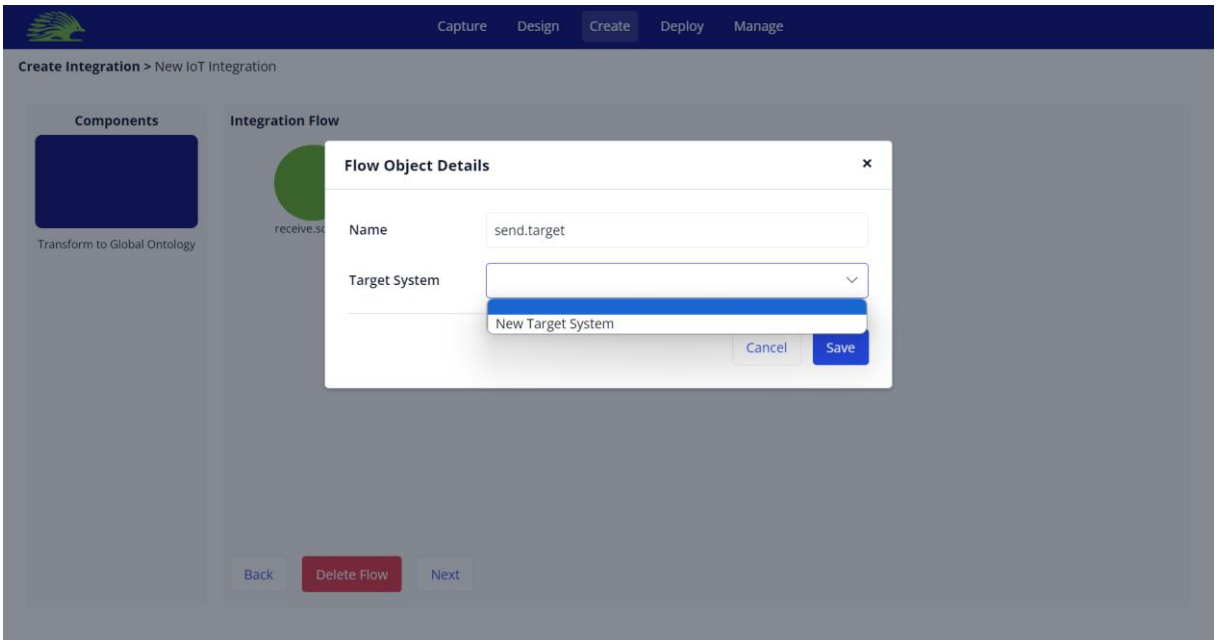

Figure 98: Selecting a Target System for an Integration Flow

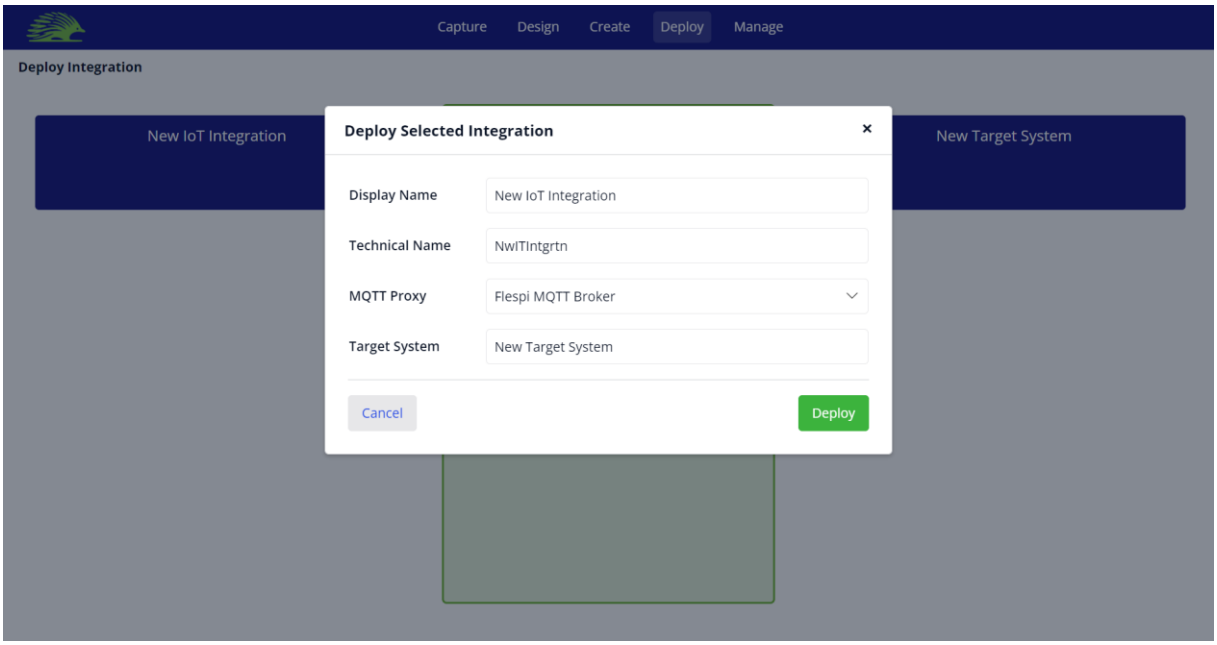

Figure 99: Deploying Integration

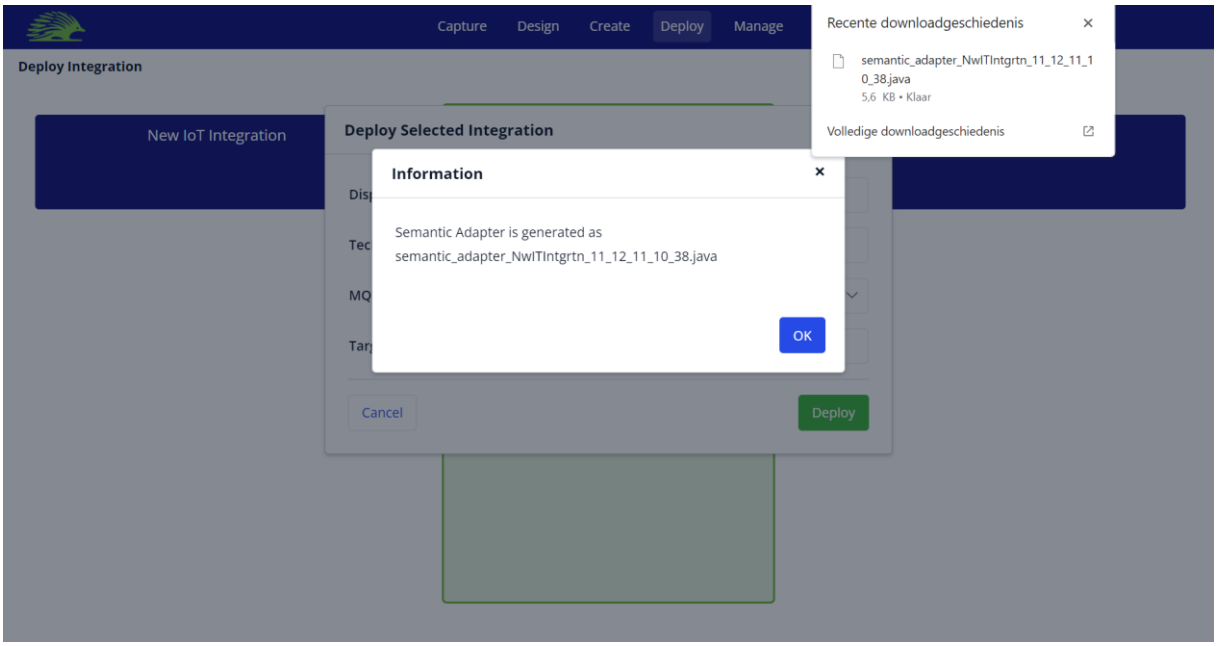

Figure 100: Generated Semantic Adapter